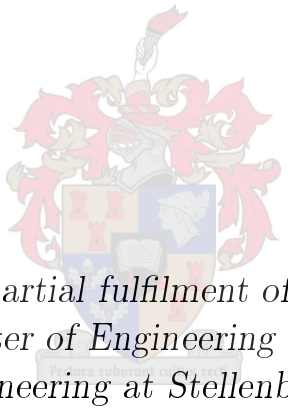
Hardware-in-Loop CubeStar Emulator

by

Armand Scholtz



Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. W.H. Steyn
Co-supervisor: Dr. L. Visagie

March 2021

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 06/02/2021 .....

Abstract

Hardware-in-Loop CubeStar Emulator

A. Scholtz

Thesis: MEng (Electronic)

February 2021

As star trackers currently provide the most accurate attitude estimation to satellites, the electronics and algorithms are constantly evolving to produce star trackers with decreased cost, size and power requirements while providing better accuracies. To perform hardware-in-loop emulations with the CubeStar in an office or laboratory environment, an emulation environment is developed and tested in this project. The emulation environment provides a more convenient alternative than the need to capture the night sky to test the efficiency of certain software algorithms on star trackers to evaluate accuracy and execution time improvements provided by upgraded electronic components and software algorithms.

The emulation environment developed in this project enables hardware-in-loop emulations to be performed with a calibrated or uncalibrated CubeStar with the use of star projection software that projects stars onto a computer monitor that simulate the real night sky. Star images are projected with average boresight angle errors of only 2.47 arcseconds and average boresight rotation errors of only 1.45 arcseconds.

The emulation environment was demonstrated by projecting stars that were both not moving or moving with constant angular rates for the CubeStar to capture. For stars that were not moving the CubeStar showed attitude estimation accuracies of boresight angular errors below 0.107° and boresight rotation errors below 0.0667°. For stars projected with angular rates of up to 0.25°/s on each axis the CubeStar showed that it could track the attitude with estimation errors of average boresight angular errors below 0.145° and average boresight rotation errors below 0.1°.

Uittreksel

Hardeware-in-Lus CubeStar Emuleerder

("Hardware-in-Loop CubeStar Emulator")

A. Scholtz

Tesis: MIng (Elektronies)

Februarie 2021

Aangesien ster volgers huidiglik die mees akkurate oriëntasie afskatting voorsien aan satelliete word beter elektronika en algoritmes konstant ontwikkel om ster volgers te vervaardig met laer koste, groottes en kragvereistes maar beter akkuraatheid. Om hardeware-in-lus emulasies met die CubeStar in 'n kantoor of laboratorium omgewing te kan uitvoer was 'n emulasie omgewing ontwikkel en getoets vir hierdie projek. Die emulasie omgewing voorsien 'n meer gerieflike alternatief as om die naghemel te moet afneem om die doeltreffendheid van sekere sagteware algoritmes op ster volgers te evalueer wat die akkuraatheid en uitvoeringstyd verbeterings sal aandui wat deur opgradeerde elektroniese komponente en algoritmes voorsien word.

Die emulasie omgewing wat in hierdie projek ontwikkel was kan hardeware-in-lus emulasies uitvoer vir 'n CubeStar wat gekalibreer is of wat nog nie gekalibreer is nie deur ster projeksie sagteware te gebruik om sterre op 'n rekenaarskerm te projekteer wat die naghemel simuleer. Ster fotos kan projekteer word met gemiddelde siglyn hoekfoute van slegs 2.47 boogsekondes en gemiddelde siglyn rotasie foute van slegs 1.45 boogsekondes.

Die emulasie omgewing was demonstreer deur sterre te projekteer wat beide nie beweeg nie of beweeg teen konstante hoeksnelhede vir die CubeStar om die sterre af te neem. Vir sterre wat nie beweeg nie het die CubeStar oriëntasie afskatting akkurathede gewys van siglyn hoekfoute onder 0.107° en siglyn rotasie foute onder 0.0667°. Met sterre wat teen konstante hoeksnelhede beweeg van tot 0.25°/s op elke as het die CubeStar gewys dat dit die oriëntasie kan volg met gemiddelde siglyn hoekfoute onder 0.145° en gemiddelde siglyn rotasie foute onder 0.1°.

Acknowledgements

I would like to thank the following people:

- My supervisor, Professor Steyn, for his guidance and for sharing his invaluable and limitless knowledge.
- My co-supervisor, Doctor Visagie, for his guidance in the software aspect of this project.
- Johanice, my parents and the rest of my family for their endless support and encouragement.
- Rina, Henry and the rest of my ESL colleagues for much needed support and advice.
- CubeSpace, for their financial support and lending a CubeStar to use in this project.
- Gabriël, for his guidance in using the CubeStar, his encouragement and his insightful advice.
- Lambert Fick & Maree Optometrists for lending an optometry lens kit and for sponsoring a convex lens.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations and Acronyms:

ADCS Attitude Determination and Control System

API Application Programming Interface

COTS Commercial off-the-shelf

CPU Central Processing Unit

DCM Direction Cosine Matrix

ECI Earth-centred Inertial

ESA European Space Agency

ESL Electronic Systems Laboratory

FoV Field of View

GUI Graphical User Interface

GVA Geometric Voting Algorithm

HIL Hardware-in-Loop

JPL Jet Propulsion Laboratory

LCD Liquid-Crystal Display

LED Light-Emitting Diode

LIS Lost-In-Space

LS Least-Squares

OLED Organic Light-Emitting Diode

P-POD Poly-Picosatellite Orbital Deployer

QUEST Quaternion Estimator

RGA Region Growing Algorithm

RMS Root mean square

TRIAD Triaxial Attitude Determination

Greek Letters:

α Right Ascension

δ Declination

ϵ Angle between ecliptic plane and equatorial plane

ϕ,θ,ψ Euler rotation angles

ω Angular rates vector

Lowercase Letters:

e Euler angle vector

f Focal length

p Rodrigues parameters

q Quaternion

r Rotation matrix element

t Intermediate reference frame vector

u Unit vector

x, y, z Axes titles

u, v Image plane axes titles

Uppercase Letters:

R Rotation matrix

K Distortion coefficient

Subscripts:

B Sensor-body coordinates

ECI Earth-centred inertial coordinates

I Inertial coordinates

M Monitor image plane coordinates

S Sensor image plane coordinates

Chapter 1

Introduction

1.1 Background

Since the CubeSat design idea was first developed in 1999 for research and education purposes, the field of nanosatellites has evolved into a large industry spanning to commercial use as well. Nanosatellites rely on sensors to feed them their attitude estimates, however, with the design constraints placed on CubeSats with respect to their size and power consumption, the accuracies provided by available sensors are limited. To increase the effectiveness of attitude estimating sensors for use on nanosatellites, constant research is performed in the field to provide technological advancements that decrease the size- and power requirements of the components used on these sensors while allowing for improved accuracies. While star trackers are the most accurate attitude estimation sensors, the development and testing of star trackers are very expensive.

A large contributor to the cost of developing star trackers is the cost of evaluating their performance, the performance testing involves expensive expeditions to remote locations where star trackers can capture stars in the night sky where certain factors caused by urban areas that effect the star tracker accuracies are reduced. Furthermore, these expeditions are time consuming, requires a lot of manpower and delays the development of the star trackers.

1.2 Project Objective

This project aims to provide a more convenient solution for the testing and verification of the onboard algorithms of a star tracker, which in turn will decrease the development and performance testing times. An emulation environment must be designed, built and tested that allows a star tracker to capture stars projected onto a monitor that simulates the real night sky. The emulation environment must be suitable to perform Hardware-in-Loop (HIL) emulations to evaluate the functioning of the star identification algorithms and attitude estimation algorithms onboard the star tracker for stars that are projected with no movement as well as stars that are projected with constant angular rates, furthermore the emulation

environment must be suitable for both a calibrated or uncalibrated star tracker.

CubeStar, a nano star tracker built by CubeSpace, was provided for use in this project. Therefore, the emulation environment must be designed with a specific focus on the use of the CubeStar. The project objectives are summarized below with a focus on the CubeStar:

1. Create a software program that can project stars onto a monitor at the correct locations with an accurate star profile similar to real captured stars.
2. Build an emulation environment sufficient for the CubeStar to capture projected stars without external factors affecting the CubeStar.
3. The surface area of the emulation environment must not exceed 1m $\times$ 1m.
4. The emulation environment must use low-budget and commercially available components.
5. The emulation environment must be suitable for both a calibrated or uncalibrated CubeStar.
6. The emulation environment must be able to project stars that are either not moving or moving with constant angular rates.
7. The emulation environment must be reliable to allow for the star identification algorithms and attitude estimation algorithms onboard the CubeStar to be successfully evaluated.

1.3 Thesis Outline

This section provides a brief overview of each chapter in this thesis.

Chapter 1: Introduction
This chapter provides a brief background to the motivation for this project and states the project objectives.

Chapter 2: Literature Study
This chapter presents the origins and purpose of CubeSats and star trackers, and thereafter the coordinate systems and attitude representations that are used in this text are discussed.

Chapter 3: Star Image Generation and Processing Algorithms
This chapter presents the software algorithms that are used in this project. The chapter is divided into three parts, star projection algorithms that are required to project stars onto a monitor to simulate the night sky, star detection and identification algorithms that are required to detect stars on a generated or captured star image and identify the stars that are on the image, and finally attitude estimation that uses the vectors from the identified stars to calculate the estimated attitude.

Chapter 4: Software Design and Analysis
In this chapter a software program written in the C♯ programming language is presented that implements the star projection algorithms to project stars onto a monitor to be captured by the CubeStar. The chapter then presents the use of star detection algorithms to analyse the generated images of projected stars to determine the accuracy to which stars are projected and the resulting attitudes that are generated.

Chapter 5: Emulation Environment Design and Analysis
This chapter presents the physical design of the emulation environment and the components that were designed and built to achieve an adequate emulation environment for the CubeStar. Firstly, the distance that the CubeStar must be placed from the monitor is discussed, followed by a discussion on the focus adjustment for the CubeStar that is focussed at infinity. Initial HIL tests performed with the CubeStar in the emulation environment are then discussed that showed the need for error correction. To correct these errors, firstly an alignment correction procedure between the CubeStar and the monitor is discussed and the results presented. Secondly, a calibration of the CubeStar in the emulation environment is discussed and the results presented, and lastly a predistortion method to predistort projected stars is discussed followed by the results.

Chapter 6: Emulation Performance Measurements
This chapter presents the final emulation performance measurements of the CubeStar in the emulation environment. Firstly, the attitude estimation accuracy of the CubeStar with still images of projected stars is discussed. Finally, the rate estimation accuracy and attitude estimation accuracy with stars that are projected with constant angular rates is discussed.

Chapter 7: Conclusions And Recommendations
This chapter presents final conclusions on the achievement of the project objectives based on the results that are presented in this thesis. Finally, some recommendations for possible improvements and future work are discussed.

Chapter 2

Literature Study

This chapter provides a brief presentation on the background of CubeSats and star trackers, and an investigation of existing star tracker emulators. The coordinate systems and attitude representations that are used in this thesis are then explained.

2.1 CubeSats

The concept of the CubeSat was first developed by Bob Twiggs and Jordi Puig-Suari to aid the trend of making satellites smaller and cheaper, and in doing so making satellite projects more accessible for educational purposes. Originally, the CubeSat nanosatellite architecture confined the size of a CubeSat to a 10x10x10cm (also referred to as 1U) cube with a mass of up to 1kg to ensure compatibility with the Poly-Picosatellite Orbital Deployer (P-POD)[3]. The P-POD is a release mechanism specifically designed for CubeSats to release the nanosatellites with a linear trajectory and minimal spin[4]. Figure 2.1 shows an example of a 1U CubeSat, the PhoneSat 2.5, that was launched into space in 2014.

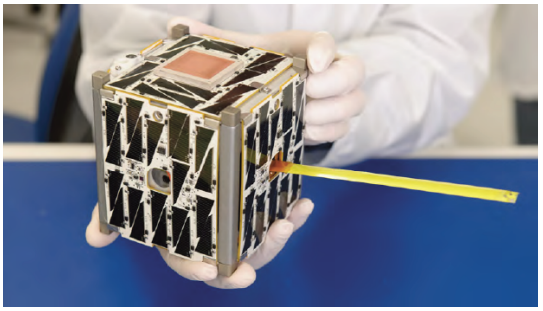


Figure 2.1: PhoneSat 2.5 [1]

Since the design specifications for CubeSats were first developed, the success of the platform launched a whole industry surrounding CubeSats for not only educational purposes, but commercial use as well. The design standard also evolved to include sizes such as the 3U, 6U and 12U CubeSats to allow for more complex CubeSat missions [5]. Figure 2.2 shows the history of the number of CubeSat launches and predictions for the amount of CubeSats to be launched in the next

5 years. The figure illustrates how large the industry is becoming and it is likely to increase even more in the next few decades.

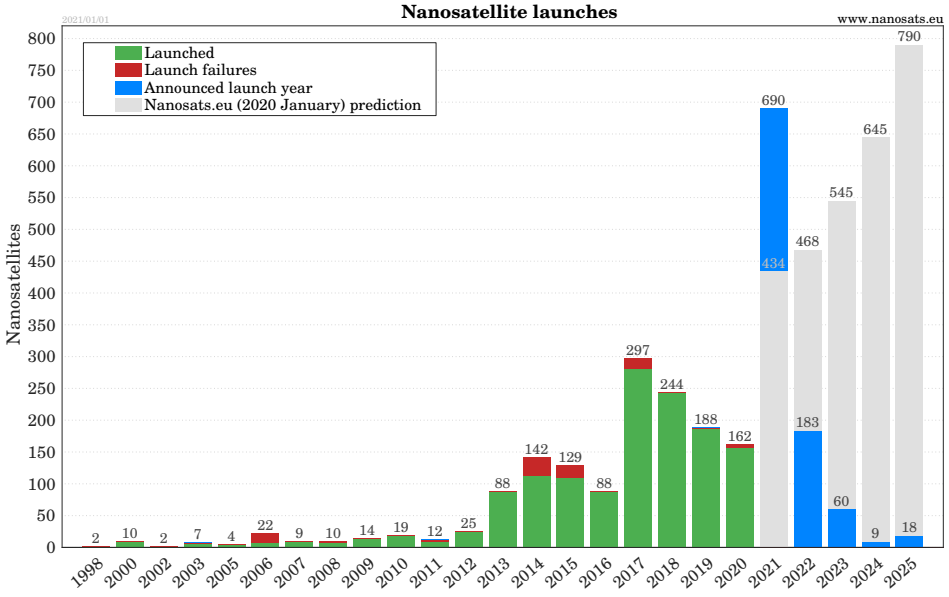


Figure 2.2: Nanosatellite Past Launches and Future Launch Predictions [2]

As the industry expands, more research goes into developing cheaper, smaller and less power consuming Commercial off-the-shelf (COTS) electronics to be used on nanosatellites. These electronics include the sensors and actuators used to determine and control the attitude of the satellites. As the requirements for CubeSats increase, the capabilities need to increase accordingly. One of the desired requirements are more accurate attitude estimation, currently star trackers are capable of producing the most accurate attitude estimation of all the sensors.

2.2 Star Trackers

Star trackers are optical instruments that use star observations to estimate the 3-axis attitude of satellites. Autonomous star trackers, such as the CubeStar, can estimate the satellite's attitude onboard and output the attitude quaternion vector to the Attitude Determination and Control System (ADCS) of the satellite without requiring any additional processing by the ADCS. To enable the star tracker to operate autonomously it uses an onboard star catalogue and complex star identification algorithms to identify the observed stars, compare them to stars in the star catalogue and then use the inertially referenced vectors of the identified stars to estimate the attitude of the satellite. There are four main aspects of star trackers that define their effectiveness, namely: their accuracy, size, mass, and power consumption. These are four very scarce resources on CubeSats and as the industry is expanding the requirement for components that use these resources more effectively is increasing.

This project is focussed on the CubeStar, a nano star tracker built by CubeSpace in Stellenbosch, South Africa. An engineering model of the CubeStar that was used in this project is shown in figure 2.3. In this case "nano" refers to the star tracker having a mass of less than 100 mg and can fit into a 1U volume CubeSat [6]. The CubeStar has a cross axis accuracy of better than 0.01° (RMS), it can identify stars up to magnitude 3.8 (resulting in a star catalogue size of 410 stars) and can output the attitude estimates at a frequency of 1 Hz [7]. The lens of the CubeStar has a horizontal Field of View (FoV) of 58° and a vertical FoV of 47°. [8].

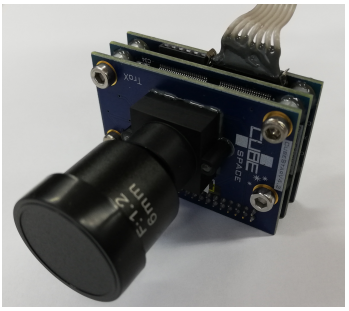


Figure 2.3: CubeStar

2.2.1 Star Catalogue

Star catalogues are lists that contain detailed specifications of the stars in the celestial space surrounding earth. In this project, the purpose of the star catalogue is twofold. Firstly, the star catalogue is required by the star projection algorithms to use the inertial star vectors in the catalogue to calculate the correct locations to where stars must be projected onto the monitor. Secondly, the star catalogue is required by the star detection algorithms to compare the calculated star body vectors from the observed stars to their inertial vectors in the catalogue to identify the stars that are observed and in turn use the identified stars to estimate the attitude of the star tracker.

The catalogue used in this project resulted from the Hipparcos satellite launched in 1989 by the European Space Agency (ESA) [9]. The mission resulted in the Hipparcos and Tycho catalogues that both contain high precision photometric and astrometric data [9]. The astrometric data includes the right ascension, declination, proper motion and visual magnitude of the stars. The right ascension and declination specifies the position of the stars in the celestial sphere surrounding earth, the proper motion describe the movement of the stars and the visual magnitude describe the brightness of the stars. There are various other data in these catalogues that are unused in this project.

2.2.2 Star Tracker Accuracy Determination

To determine the accuracy of a star tracker before it is launched on a satellite the most accurate method of choice remains to capture stars in the actual night

sky. This method, however, is very inconvenient as expensive and time-consuming outdoor expeditions are necessary due to various factors that reduce the accuracy of night sky observations. These factors include light pollution, astronomical observation and atmospheric refraction [10]. Due to the pollution caused by cities, these factors are worse close to cities and therefore locations far from urban areas are required to perform the performance measurements of star trackers. To perform these tests at remote locations dedicated hardware setups are required, which further increases the cost and time of such expeditions. Although this method is used to determine the final performance measurements of a star tracker and to test the functioning of the algorithms onboard, research such as this project is investigating alternative methods that can provide a more convenient method that can be implemented indoors in an office or laboratory setting [10],[11].

Two existing methods that addresses this problem are discussed, the first method is to use collimated light sources to represent stars for the star tracker to capture, the second method is the use of an emulation environment, such as the one discussed in this project, to simulate the night sky for the star tracker to capture.

The first method consist of using a collimated light source to represent a single star [10]. A drawback of this method is that for a constellation of stars to be created a collimated light source for each star will be required and therefore the physical environment for the stars to be placed in can become very large and complex. Another drawback is that for angular rate emulations to be performed the star tracker would need to be placed on a reliable rate table that can rotate the star tracker around an axis, rate tables are incredibly expensive and not easily accessible.

The second method is an emulation environment where stars are displayed onto a monitor at their real locations to be captured by a star tracker. The emulation environment will typically consist of a platform for the star tracker to be placed on and a monitor on which the stars can be projected, the components must be placed in a dark environment where there are no additional light sources other than the light coming from the projected stars. Although the accuracies that are achievable through this method is not sufficient to be able to determine the final performance measurements of a star tracker, it is a much more convenient option for testing the functioning of certain algorithms of the star tracker rather than the need to capture the night sky. The next section investigates existing star emulators that are found in literature.

2.2.3 Existing Star Tracker Emulators

Existing star emulators that are used to test the functionality of a star tracker are investigated in this section.

The first article that will be investigated is that of Rufino et al. [11]. A large and high resolution monitor is used for projected stars and a collimator lens is

used to ensure that the projected stars appear at an infinite distance from the star tracker, this lens has a focal length of 1.3m and must therefore be placed at this distance from the monitor. The star tracker is placed on high-precision translation stages that allow for extremely fine adjustments of the star tracker and collimator to ensure that they are aligned properly with the monitor. Furthermore, the entire emulation setup is placed on a bench with pneumatic vibration isolation for alignment stability between the star tracker and the monitor.

The second article that will be investigated is that of Roshanian et al. [12] who also presents a star tracker emulation environment, however since the star tracker emulator is not the topic of this article the emulator itself is not discussed in detail. However, it is concluded that the environment is quite compact. Stars are projected on a small Liquid-Crystal Display (LCD) monitor and the star tracker is placed at a very short distance from the monitor. This short distance is possible since the star tracker that is used has a FoV of only 8°, which is significantly smaller than the 58° horizontal FoV of the CubeStar, as mentioned in section 2.2. Furthermore, a collimator is also used to allow the star tracker to focus on the small LCD monitor. The star tracker is also placed on a mechanical setup that enables fine adjustments of the star tracker and collimator.

The two articles discussed above show that there is a trend in existing star tracker emulators to be fairly expensive as they use expensive components. In both cases the star tracker uses an expensive lens collimator and in both cases the star trackers are placed on expensive translation stages that allow very fine adjustments.

The last article that was investigated is that of Samaan et al. [13]. This article took an entirely different approach to construct a star tracker emulator by using a Jena-Optronik Optical Sky Field Simulator (OSI). The Jena-Optronik OSI is a device that provides a connection for a star tracker to be mounted directly onto the device with stars that are projected onto a micro display inside of the OSI, a collimated optical head ensures that stars are observed at an infinite distance. This article demonstrates that there are commercial options available to the problem addressed by this project, however since the Jena-Optronik OSI is a very specialised piece of equipment it is likely to be extremely expensive and not affordable for most companies that produce star trackers.

2.3 Coordinate Systems

Coordinate systems are used as a reference frame to describe the positions of objects relative to this reference frame, as well as to describe the attitude of objects within the reference frame. In this project there are 4 important coordinate systems, namely Earth-centred Inertial (ECI) coordinates, Sensor Image Plane coordinates, Monitor Image Plane coordinates and Sensor-Body coordinates. The four coordinates systems are discussed in this section.

2.3.1 Earth-centred Inertial Coordinates

This coordinate system, also referred to as inertial coordinates, is important as it is used to describe the position and motion of celestial objects, such as stars, relative to the earth. Figure 2.4 provides a visual representation of the ECI coordinate frame to aid the discussion that follows.

The origin of the ECI frame lies at the centre of mass of the earth with the z-axis of the frame normal to the earth's equatorial plane and points in the direction of the earth's North pole. The z-axis therefore lies collinear with the earth's spin axis. The earth's equatorial plane is normal to the earth's spin axis while the ecliptic plane is the centre of the earth as it orbits the sun, the ecliptic plane lies at an angle of $\epsilon = 23.4^{\circ}$ from the earth's equatorial plane [14]. At a specific point in space a vector can be drawn from the centre of mass of the earth to the sun that lies collinear with where the ecliptic plane intersects the equatorial plane, this vector is called the Vernal Equinox and the ECI frame's x-axis lies collinear to this vector. The y-axis is chosen to complete the right-handed coordinate system. Although the ECI frame is aligned with the earth's spin axis, the ECI frame remains fixed and does not rotate with the earth.

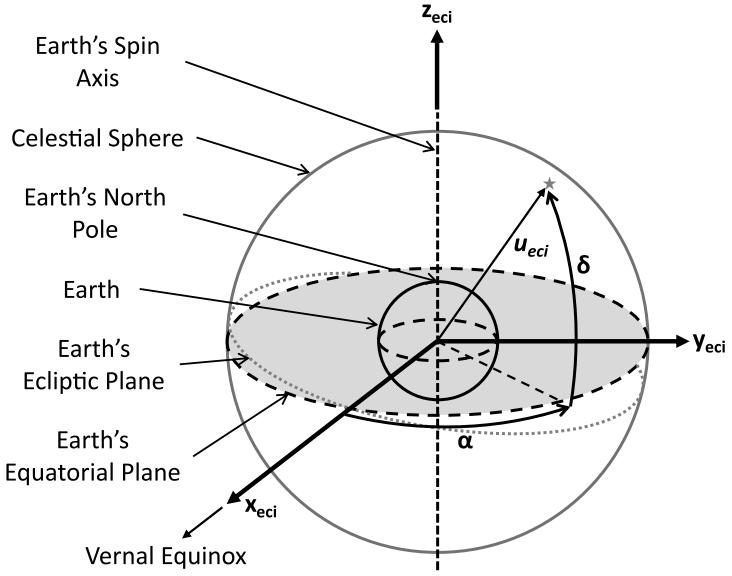


Figure 2.4: Earth-centred Inertial Coordinate Frame

Due to the earth's spin axis precessing over time, the ECI frame moves relative to the stars. For this reason, a standard ECI frame, called the J2000 ECI coordinate system, is used as a time reference and refers to the Vernal Equinox at 12:00 on 1 January 2000.

As illustrated in figure 2.4, ECI coordinates are often given as right ascension (α), in angular hours, minutes and seconds, and declination (δ), in sexagesimal (numbers with 60 as its base) degrees, minutes and seconds. To convert these

values to decimal degrees, equations 2.1 and 2.2 can be used.

$$\alpha_{degrees} = (Hour + \frac{Minute}{60} + \frac{Second}{3600}) \cdot 15 \tag{2.1}$$

$$\delta_{degrees} = (Degrees + \frac{Minute}{60} + \frac{Second}{3600}) \tag{2.2}$$

where

$\alpha_{degrees}$: right ascension in decimal degrees
$\delta_{degrees}$: declination in decimal degrees

The components of an ECI coordinate unit vector, $\boldsymbol{u_{eci}}$, can then be calculated with equations 2.4 to 2.6.

$$\boldsymbol{u_{eci}} = \begin{bmatrix} x_{eci} & y_{eci} & z_{eci} \end{bmatrix}^T \tag{2.3}$$

with

$$x_{eci} = \cos(\delta)\cos(\alpha) \tag{2.4}$$

$$y_{eci} = \cos(\delta)\sin(\alpha) \tag{2.5}$$

$$z_{eci} = \sin(\delta) \tag{2.6}$$

where:

x_{eci}: x-axis in ECI coordinates
y_{eci}: y-axis in ECI coordinates
z_{eci}: z-axis in ECI coordinates

2.3.2 Sensor Image Plane Coordinates

This coordinate system is a two-dimensional reference frame that describe the position of a pixel on the image plane of the sensor. The origin (point (0,0)) of this system, when the sensor is viewed from the front, is the top right corner, as shown in figure 2.5 below [8]. The v-axis lies vertical and increases downward and the u-axis lies horizontal and increases to the left.

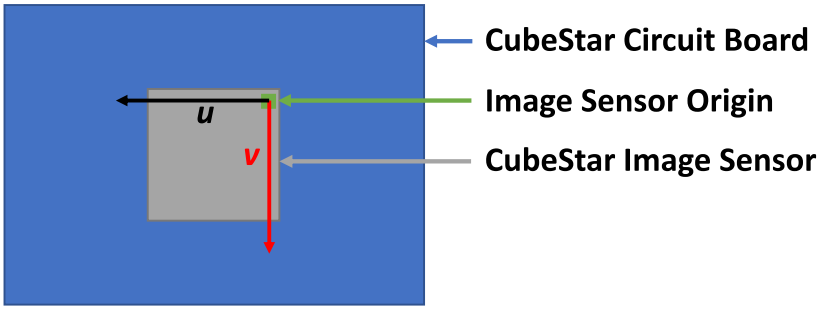


Figure 2.5: CubeStar Sensor Image Plane Coordinates (when viewed from the front)

2.3.3 Monitor Image Plane Coordinates

This coordinate system is used to describe the location of a projected pixel on the monitor. This coordinate system is very similar to sensor image plane coordinates, however the origin (point (0,0)) of the monitor image plane coordinates lies at the top left corner when viewed from the front. Therefore the x-axis, while still lying horizontal, now increases to the right. The y-axis still lies vertical and increases downward. Figure 2.6 shows a visual representation of the monitor image plane coordinates.

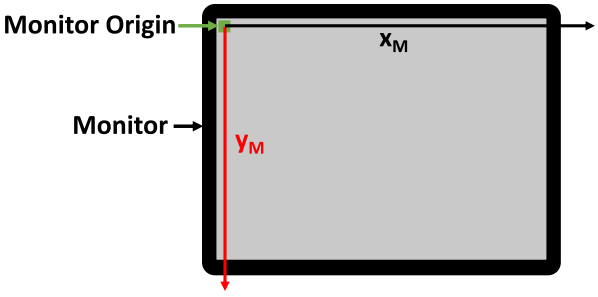


Figure 2.6: Monitor Image Plane Coordinates (when viewed from the front)

2.3.4 Sensor-Body Coordinates

This coordinate system, also referred to as body coordinates, is a three-dimensional reference frame with its origin at the focal point of the lens [8]. This focal point is defined as the point whereto light rays entering the lens that travels parallel to the principal axis of the lens will converge. The axes of this coordinate system are chosen to correspond with the body coordinate system of the CubeStar as described in the interface control document [8]. The positive z-axis points towards the boresight of the lens and therefore the direction that the sensor is facing, the x-axis lies collinear to the image sensor's x-axis but points in the opposite direction and the y-axis is chosen to complete the right-handed coordinate system. The sensor-body coordinate system is visually represented in figure 2.7 below.

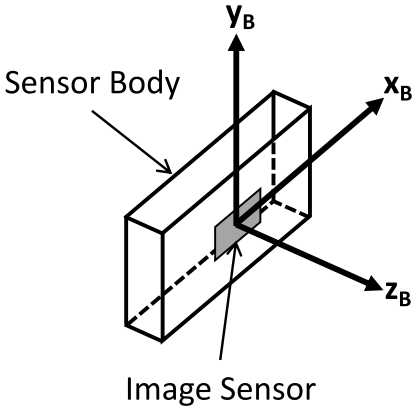


Figure 2.7: Sensor-Body Coordinates

2.4 Attitude Representations

This section discusses the techniques used to represent the attitude of objects in three dimensional space. Not only the vector direction pointing to an object is required, but also the rotation about this direction. Several attitude representation techniques exist, of which the three that is used in this project will be discussed. These include rotation matrices, Euler angles and quaternions.

2.4.1 Rotation Matrix

A rotation matrix, also referred to as an attitude matrix or Direction Cosine Matrix (DCM), is an orthogonal 3x3 matrix that describes the rotation of a vector. When such a matrix is multiplied with a vector, the vector is rotated in three-dimensional space while its length is preserved. The elements of a general rotation matrix has the form described in equation 2.7.

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.7}$$

For rigid-body transformations the determinant of the rotation matrix will be equal to 1 [15]. Considering two reference frames, an inertial reference frame and a body reference frame, a rotation matrix $\boldsymbol{R}_I^B$ defines the rotation from the inertial to the body reference frame. Thus, a vector in the inertial reference frame multiplied by the rotation matrix $\boldsymbol{R}_I^B$ will result in that vector represented in the body reference frame, such that:

$$\boldsymbol{v}_B = \boldsymbol{R}_I^B \boldsymbol{v}_I \tag{2.8}$$

2.4.2 Euler Angles

Euler Angles is an alternative method of describing the attitude of an object and consist of three consecutive rotations about three axes in sequence. The three rotation angles that describe the rotation about each axis in the sequence can be represented as an Euler angle vector where the elements of the vector are referred to as the Euler angles such that

$$\boldsymbol{e}_{angles} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T \tag{2.9}$$

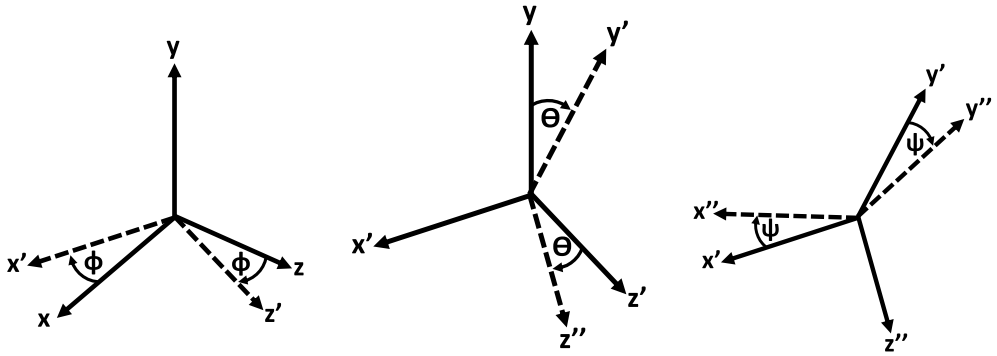
Several rotation sequences exist for different applications, as the Euler angles suffer from mathematical singularities at various locations. CubeSpace uses the 2-1-3 (also known as Y-X-Z) rotation sequence and therefore this sequence is the chosen rotation sequence for this project as well. The rotation of each axis is described by equations 2.10, 2.11 and 2.12.

$$\boldsymbol{R}_2(\phi) = \begin{bmatrix} \cos(\phi) & 0 & -\sin(\phi) \\ 0 & 1 & 0 \\ \sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \tag{2.10}$$

$$\boldsymbol{R}_1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.11}$$

$$\boldsymbol{R}_3(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

Figure 2.8 shows the three consecutive rotations that form a 2-1-3 rotation sequence. The first rotation, $\boldsymbol{R}_2(\phi)$, is around the y-axis with the angle ϕ and is shown in figure 2.8a. The resulting axes of the first rotation are x' and z'. The second rotation, $\boldsymbol{R}_1(\theta)$, will be around the resulting x-axis from the first rotation, x', with the angle θ. The second rotation is shown in figure 2.8b, the resulting axes are y' and z''. The third and last rotation, $\boldsymbol{R}_3(\psi)$, will be around the resulting z-axis from the second rotation, z'', with the angle ψ. This rotation is shown in figure 2.8c and will have resulting axes x'' and y''.



(a) Euler rotation around y-axis with angle ϕ (b) Euler rotation around x'-axis with angle θ (c) Euler rotation around z"-axis with angle ψ

Figure 2.8: Euler Rotation Sequence 2-1-3

The full rotation sequence can be described by a single rotation matrix $\boldsymbol{R}_{213}(\phi, \theta, \psi)$, which is shown in equation 2.13 below. Note that for a more compact notation in equation 2.13 the sine and cosine trigonometric functions are written as "s" and "c" respectively.

$$\begin{aligned} \boldsymbol{R}_{213}(\phi, \theta, \psi) &= \boldsymbol{R}_2(\phi)\boldsymbol{R}_1(\theta)\boldsymbol{R}_3(\psi) \\ &= \begin{bmatrix} c(\phi)c(\psi) - s(\phi)s(\theta)s(\psi) & c(\phi)s(\psi) + s(\phi)s(\theta)c(\psi) & -s(\phi)c(\theta) \\ -c(\theta)s(\psi) & c(\theta)c(\psi) & s(\theta) \\ s(\phi)c(\psi) + c(\phi)s(\theta)s(\psi) & s(\phi)s(\psi) - c(\phi)s(\theta)c(\psi) & -c(\phi)c(\theta) \end{bmatrix} \end{aligned} \tag{2.13}$$

2.4.3 Quaternions

The last attitude representation that will be discussed is quaternions, first proposed by William Hamilton in 1843. It provides a more compact representation than rotation matrices and are more stable than Euler angles as it does not suffer from singularities. There are two quaternion vector notations that can be used, the form Hamilton first proposed and the form proposed by Jet Propulsion Laboratory (JPL) [16]. The CubeStar uses the JPL notation, therefore this was the chosen notation for this project. The JPL vector quaternion form is described by equation 2.14.

$$\boldsymbol{q} = q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} + q_4 \tag{2.14}$$

Quaternions can also be described as the sum of a scalar part, q_4, and an imaginary vector part, $\boldsymbol{q}_v = q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$, and can therefore be written as

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_v \\ q_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{2.15}$$

When quaternions are used to describe rotations, they can be referred to as rotation quaternions. A rotation quaternion is used to describe a unit vector,$\boldsymbol{u}$, in a coordinate frame and a rotation, θ, about this unit vector. The rotation quaternion then describes the relationship between two three-dimensional vectors. When used in this manner, the quaternion unit vector can be described as

$$\boldsymbol{q}_v = \boldsymbol{u}\sin(\theta/2) \tag{2.16}$$

$$q_4 = \cos(\theta/2) \tag{2.17}$$

with

$$\boldsymbol{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T \tag{2.18}$$

Conversions Between Quaternions and Rotation Matrices:

A quaternion $\boldsymbol{q}$ in the form described by equation 2.15 can be converted to its corresponding rotation matrix $\boldsymbol{R}$ with equation 2.19 below.

$$\boldsymbol{R}(\boldsymbol{q}) = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \tag{2.19}$$

A rotation matrix $\boldsymbol{R}$ in the form described by equation 2.7 can be converted to its corresponding rotation quaternion $\boldsymbol{q}$ with equation 2.20 below.

$$\begin{aligned} q_1 &= \frac{(r_{23} - r_{32})}{4q_4} \\ q_2 &= \frac{(r_{31} - r_{13})}{4q_4} \\ q_3 &= \frac{(r_{12} - r_{21})}{4q_4} \\ q_4 &= \frac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1} \end{aligned} \tag{2.20}$$

Equation 2.20 is only valid for cases where $q_4 \neq 0$ or $\theta \neq 180°$, the equations for other cases to convert a rotation matrix to its corresponding quaternion are discussed in detail by Diebel in [15].

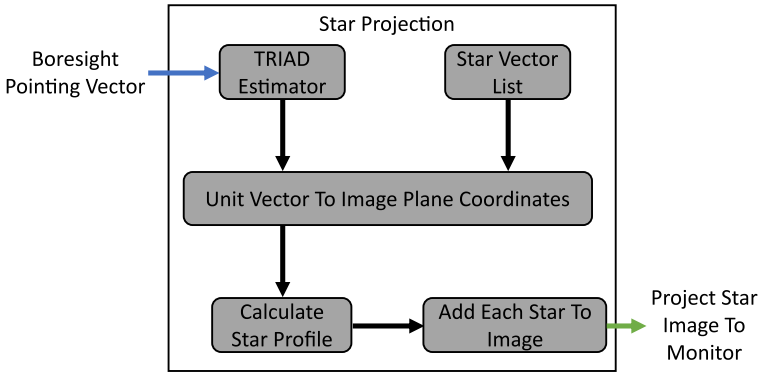
2.5 Chapter Summary

The purpose of this chapter was to provide background information on CubeSats, star trackers and existing star tracker emulators. This chapter also discussed important concepts regarding different coordinate systems and attitude representations as they will be used in the design of the software algorithms and results analysis in the coming chapters.

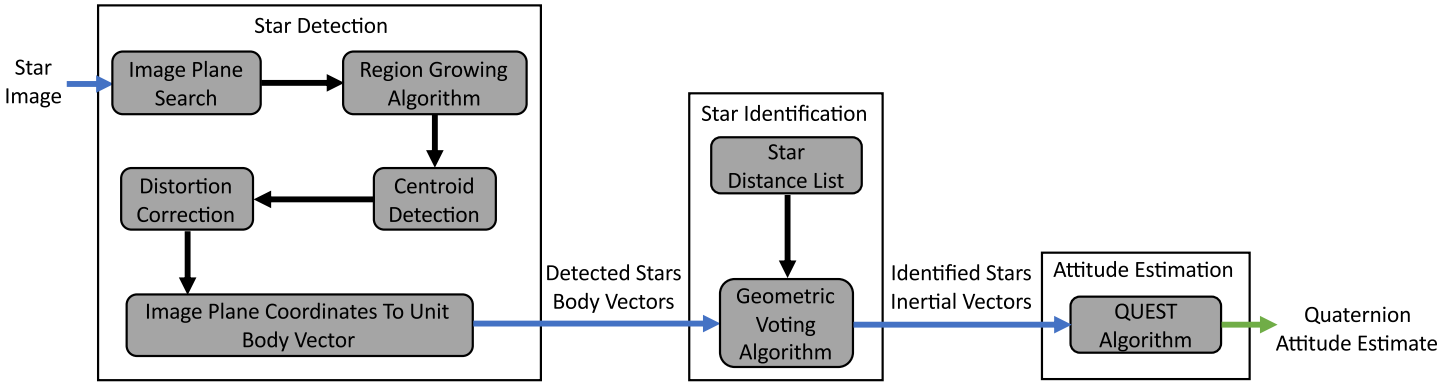
Chapter 3

Star Image Generation and Processing Algorithms

The algorithms and techniques that are used in this project are divided into three parts: 1) star projection, 2) star detection and identification and 3) attitude estimation. Figure 3.1a provides an overview of the star projection algorithms and figure 3.1b provides an overview of the star detection and identification algorithms as well as the attitude estimation algorithms.



(a) Star Projection Algorithms Overview



(b) Star Detection and Identification- and Attitude Estimation Algorithms Overview

Figure 3.1: Star Image Generation and Processing Algorithms Overview

The star projection section discusses all the algorithms and techniques that are required to project stars onto a monitor at the correct locations and with accurate star profiles similar to actual stars captured in the night sky. The two prerequisites to calculate the image plane coordinates of the stars are the inertial vectors of the stars and a boresight pointing vector of the star tracker to calculate the attitude. Therefore, the first section starts by discussing a star vector list that contains the inertial vectors of all the stars in the star catalogue and a Triaxial Attitude Determination (TRIAD) estimator that is used to generate an initial attitude given a boresight pointing vector. The method used to calculate the body vector of each star is then discussed, followed by the calculation of the image plane coordinates of a star to find the location that a star must be projected to. Lastly, the technique used to generate the unique star profile of each star to accurately simulate a star in the night sky is discussed.

The star detection and identification section discusses all the techniques and algorithms that are required to analyse generated or captured star images. Firstly, the algorithms used to find the star centroids are presented, this includes the Image Plane Search-, Region Growing- and Centroiding algorithms. Due to the presence of distortion in the lens of the CubeStar, a distortion correction method is then presented. To achieve star identification efficiently, the generation of a star distance list from the star catalogue is discussed after which the Geometric Voting Algorithm is explained that will match the stars to their corresponding catalogue counterparts.

The final section of this chapter, attitude estimation, presents different methods for attitude estimation and explains why the Quaternion Estimator (QUEST) algorithm was the chosen method to estimate the attitude of the star tracker. The final discussion in this section is the method used to calculate the errors between two attitudes.

3.1 Star Projection

The star projection algorithms are mainly used by the star projection software created for this project (discussed in chapter 4). To project stars onto a monitor, firstly a list containing all of the inertial vectors of the stars that could be projected is required. From there, various techniques are used to project the correct stars in the list to the correct locations on the monitor with an accurate representation of what a star tracker will observe when imaging real stars in the night sky.

3.1.1 Star Vector List

The star vector list contains the inertial vectors of stars that the software requires to determine where on the image a star should be projected. A subset of the Hipparcos catalogue, discussed in section 2.2.1, was obtained from the online star catalogue database "VizieR" [17]. From this subset the star identification numbers (HIP), right ascension (α in $Hours$: $minutes$: $seconds$ units), declination (δ

in *degrees* : *minutes* : *seconds* units) and visual magnitude (M_v) were extracted for the stars up to a magnitude of $3.8M_v$ and sorted by decreasing brightness (increasing visual magnitude M_v). The right ascension was first converted from *Hours* : *minutes* : *seconds* units to degrees using equation 2.1, and the declination from *degrees* : *minutes* : *seconds* units to degrees using equation 2.2. The right ascension and declination was then used to calculate the 3-axis unit vector in ECI coordinates of each star using equations 2.4 to 2.6. The resulting list contains 410 stars identified by their identification number HIP from the star catalogue, they were also given a second identification number, $CSID$, which is the same identification number for each respective star that the CubeStar's internal star list uses [18]. An extract of the star vector list is given in table 3.1.

Table 3.1: Star Vector List

HIP	x_{eci}	y_{eci}	z_{eci}	M_v	δ	α	CSID
32349	-0.1874	0.939197	-0.28773	-1.44	-0.29186	1.767745	113
30438	-0.06323	0.602743	-0.79543	-0.62	-0.91971	1.675309	108
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99675	0.37732	-0.57206	0.728267	3.8	0.81579	5.295468	361

3.1.2 Initial Attitude Calculation

Before any stars can be projected onto the monitor image plane, an attitude is required to emulate where the star tracker's boresight is pointing to in the night sky. As discussed later in the text (chapter 4), for ease of use during emulations any star in the catalogue can be chosen as the boresight pointing star at the start-up of the software. The initial attitude must therefore be generated as the rotation matrix that will result in the specified pointing star lying in the middle of the monitor image plane. Therefore, the inertial vector of the selected star is used as the inertial boresight pointing vector and a TRIAD estimator is used to generate the corresponding rotation matrix.

3.1.2.1 TRIAD Estimator Background

The TRIAD Estimator is an algorithm that was invented by Harold D. Black in 1964 [19] that uses two pairs of vector observations to calculate the 3-axis attitude of a satellite. Two measured vectors are given in both satellite body coordinates ($\boldsymbol{v}_{1b}$ and $\boldsymbol{v}_{2b}$) and inertial coordinates ($\boldsymbol{v}_{1i}$ and $\boldsymbol{v}_{2i}$) and used to construct the components for an intermediate reference frame ($\boldsymbol{t}_1$, $\boldsymbol{t}_2$ and $\boldsymbol{t}_3$). The components of the intermediate reference frame forms two rotation matrices, $\boldsymbol{R}_T^B$ and $\boldsymbol{R}_I^T$, which when multiplied together forms the attitude DCM $\boldsymbol{R}_I^B$ such that

$$\boldsymbol{R}_I^B = \boldsymbol{R}_T^B \boldsymbol{R}_I^T \tag{3.1}$$

The measurement assumed to be the most accurate is used as the first component of the intermediate reference frame such that

$$\boldsymbol{t}_{1I} = \boldsymbol{v}_{1I} \quad ; \quad \boldsymbol{t}_{1B} = \boldsymbol{v}_{1B} \tag{3.2}$$

The second component for the intermediate reference frame is chosen to be orthogonal to the two measured vectors such that

$$\boldsymbol{t}_{2I} = \frac{\boldsymbol{v}_{1I} \times \boldsymbol{v}_{2I}}{|\boldsymbol{v}_{1I} \times \boldsymbol{v}_{2I}|} \quad ; \quad \boldsymbol{t}_{2B} = \frac{\boldsymbol{v}_{1B} \times \boldsymbol{v}_{2B}}{|\boldsymbol{v}_{1B} \times \boldsymbol{v}_{2B}|} \tag{3.3}$$

The third and last component of the intermediate reference frame is calculated to be orthogonal to the first two components such that

$$\boldsymbol{t}_{3I} = \boldsymbol{t}_{1I} \times \boldsymbol{t}_{2I} \quad ; \quad \boldsymbol{t}_{3B} = \boldsymbol{t}_{1B} \times \boldsymbol{t}_{2B} \tag{3.4}$$

The two rotation matrices describing the orientation relative to the intermediate frame can then be constructed from the three intermediate vector components such that

$$\boldsymbol{R}_T^B = \begin{bmatrix} \boldsymbol{t}_{1B} & \boldsymbol{t}_{2B} & \boldsymbol{t}_{3B} \end{bmatrix} \tag{3.5}$$

and

$$\boldsymbol{R}_I^T = \begin{bmatrix} \boldsymbol{t}_{1I} & \boldsymbol{t}_{2I} & \boldsymbol{t}_{3I} \end{bmatrix}^T \tag{3.6}$$

The attitude rotation matrix $\boldsymbol{R}_I^B$ can then be calculated with equation 3.1.

3.1.2.2 TRIAD Estimator Practical Use

The pointing star's inertial unit vector as well as the right ascension and declination in degrees is extracted from the star vector list. Since the star tracker's sensor body coordinates has its boresight as the z-axis (see section 2.3.4), the first body vector is initialized to

$$\boldsymbol{v}_{1B} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

and the first inertial vector is set to the pointing star's inertial unit vector components such that

$$\boldsymbol{v}_{1I} = \begin{bmatrix} x_{eci} & y_{eci} & z_{eci} \end{bmatrix} \tag{3.8}$$

To ensure that the rotation matrix is generated with the pointing star in the centre of the monitor plane, the second body vector is initialized to

$$\boldsymbol{v}_{2B} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \tag{3.9}$$

and with equations 2.4 to 2.6, the pointing star's right ascension and declination is used to calculate the second inertial vector with a value of 90° added to the declination such that

$$\boldsymbol{v}_{2I} = \begin{bmatrix} \cos(\delta + 90°)\cos(\alpha) & \cos(\delta + 90°)\sin(\alpha) & \sin(\delta + 90°) \end{bmatrix} \tag{3.10}$$

Equations 3.9 and 3.10 will ensure that the second body vector lies orthogonal to the first body vector, and that the second inertial vector lies orthogonal to the first inertial vector. The two vector pairs are then sent to the TRIAD estimator to calculate the rotation matrix with the method described in the previous section. The pseudocode for this algorithm as it can be used in the software can be viewed

in Algorithm 1.

Algorithm 1: Generate Attitude Matrix With TRIAD Estimator

input : $pointingStar$, $starsList$
output: R_{ib}

declare $v1B = [0 \quad 0 \quad 1]$
declare $v2B = [0 \quad 1 \quad 0]$

$index1$ = find index of $pointingStar$ in $starList$
if $index1 < 0$ **then**
  return "Invalid Star"
end

ra = right ascension from $starsList(index1)$
dec = declination from $starsList(index1)$
$dec2 = dec + \frac{\pi}{2}$

$v1N$ = use equations 2.4 to 2.6 with (ra, dec) and normalize
$v2N$ = use equations 2.4 to 2.6 with (ra, $dec2$) and normalize

$t1B = v1B$
$t2B$ = `crossProduct(`$v1B$, $v2B$`)` and normalize
$t3B$ = `crossProduct(`$t1B$, $t2B$`)` and normalize

$t1N = v1N$
$t2N$ = `crossProduct(`$v1N$, $v2N$`)` and normalize
$t3N$ = `crossProduct(`$t1N$, $t2N$`)` and normalize

$Rbt = [t1B \quad t2B \quad t3B]$
$Rti = [t1N \quad t2N \quad t3N]'$
$Rbi = Rbt * Rti$

return Rbi

3.1.3 Unit Vector To Image Plane

To calculate where on the monitor image plane a star centroid will be, the common problem of projecting a three dimensional "world" point onto a two-dimensional image plane must be solved. The CubeStar has a narrow enough FoV lens for use of a simple pinhole camera model, as shown in figure 3.2. In figure 3.2 the two-dimensional image plane axes are u and v and the three-dimensional world axes that form a left-handed coordinate frame are x, y and z. The origin of the world axes lies at the centre of the pinhole and the origin of the image plane (u_o, v_o) lies in the centre of it, the image plane lies a distance of f away from the pinhole which is known as the focal length.

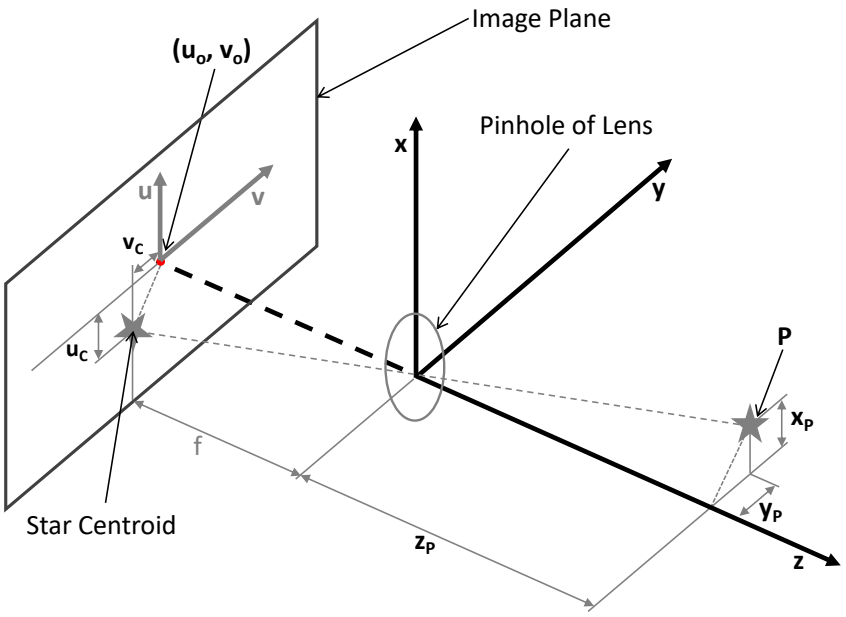


Figure 3.2: The Pinhole Camera Model

From the geometry of similar triangles it follows that

$$\frac{u_c}{f} = \frac{x_P}{z_P} \quad , \quad \frac{v_c}{f} = \frac{y_P}{z_P} \tag{3.11}$$

which simplifies to

$$u_c = \frac{f x_P}{z_P} \quad , \quad v_c = \frac{f y_P}{z_P} \tag{3.12}$$

As shown with the star in figure 3.2, due to the optics of the pinhole camera model an observed object will lie at an inverted position on the image plane. Since the algorithms onboard the CubeStar already compensates for the inverted image, the projected image must not be projected with this inverted orientation but instead the object must be projected with its real world orientation. Therefore, equation 3.12 is modified with a reversed sign to invert the image such that

$$u_c = \frac{-f x_P}{z_P} \quad , \quad v_c = \frac{-f y_P}{z_P} \tag{3.13}$$

Another modification to equation 3.13 is required since the origin of the image plane in figure 3.2 lies at the centre of the image whereas the origin of the CubeStar's sensor plane lies at the top right, as discussed in section 2.3.2. This is solved by adding half of the image width to the u-axis and half of the image height to the v-axis, such that

$$u_C = \frac{-f x_P}{z_P} + \frac{ImageWidth}{2} \quad , \quad v_C = \frac{-f y_P}{z_P} + \frac{ImageHeight}{2} \tag{3.14}$$

With the above equations, a list can be created that contains all of the star centroids that must be projected with their centroid locations. The pseudocode for this step is shown in Algorithm 2. Note that in Algorithm 2, the function

"preDistort()" is first discussed in section 5.2.5.

Algorithm 2: Calculate Star Centroids

input : $attitudeMatrix$, $starsList$
output: $starsToDraw$

for $j = 1$ *to* $j = len$ *(where len is length of starsList)* **do**
  get inertial vector of current star, $vstari(j)$, from $starsList(j)$
  $vx = attitudeMatrix(1,:) * vstari(j)$ (from equation 2.8)
  $vy = attitudeMatrix(2,:) * vstari(j)$ (from equation 2.8)
  $vz = attitudeMatrix(3,:) * vstari(j)$ (from equation 2.8)
  if $vz < 0$ **then**
    skip star j
  end
  $cx = -1 * focalLength * \frac{vx}{vz} + \frac{imageWidth}{2}$ (from equation 3.14)
  $cy = -1 * focalLength * \frac{vy}{vz} + \frac{imageHeight}{2}$ (from equation 3.14)
  if *cx AND cy lies in image window* **then**
    if *preDistortFlag is true* **then**
      $(cxnew, cynew)$ = `preDistort(`cx, cy`)`
      add $(cxnew, cynew)$ to $starsToDraw$ list
    else
      add (cx, cy) to $starsToDraw$ list
    end
  end
end
return $starsToDraw$

3.1.4 Projected Star Profile

For project objective 1 to be satisfied, projected stars must have accurate profiles to represent actual observed stars in the night sky. The method to achieve this is discussed in this section.

Stars can be considered a point light source, so with perfectly focused optics a star will illuminate only a couple of pixels on an image plane, depending on the brightness and size of the star. Star centroiding algorithms that determine the centroid of a star relies on the intensity of each pixel belonging to a star to achieve what is known as sub-pixel accuracy, meaning that the centroid can be determined to where in a pixel the centroid lies. Contrary to intuition, perfectly focused optics causes the centroiding algorithms to be less accurate as the stars occupy very few pixels to be used in the algorithms. Therefore, star tracker optics are commonly defocused slightly from perfect focus in order to blur stars so that it occupies more pixels, which in turn allows for more accurate star centroid determination.

As Greyling discussed extensively in his work [20], the spread of the light coming from stars is very complex, but a star can accurately enough be modelled using a 2D Gaussian distribution for the profile of a star.

3.1.4.1 Gaussian Distributed Star Profile

Since the goal of the software includes simulating relative motion between the stars and the star tracker in real-time (not just stationary stars) by projecting the stars with constant angular rates, the refresh rate of the star projection software must be high enough to allow the projected stars to be updated faster than the CubeStar can perform star detection. For each star that lies in the current FoV, the intensity of each pixel belonging to that star must be calculated. Using the standard two-dimensional Gaussian distribution function (see equation 3.15) for each pixel will involve a lot of calculations to be performed by the software and therefore will take a lot of processing time, so another solution to this problem was explored in this project.

A star's position on the projected image is determined with sub-pixel accuracy from the star catalogue with equation 3.14, the stars must therefore have an accurate pixel intensity profile, referred to as the "kernel" of the star, to enable the projected stars to be identified at the sub-pixel location where they are projected. Each star will have unique pixel intensity profiles depending on where in a pixel the star's centroid is located. Using equation 3.15, a lookup table is generated that contains the values of a two-dimensional Gaussian distribution that is used for both the x- and y-axis.

$$f_G(x) = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma^2}} \cdot e^{\frac{-(x-a)^2}{2 \cdot \sigma^2}} \tag{3.15}$$

where
σ: standard deviation of the distribution
a: mean of the distribution

The Gaussian distributed lookup table is dependant on three parameters, referred to as the Gaussian lookup table parameters, namely:

1. The size of the lookup table.
2. The standard deviation σ of the Gaussian distribution for equation 3.15.
3. The resolution to which the lookup table is generated.

The size of the lookup table is determined by both the dimensions chosen for the star kernels as well as the chosen resolution for the lookup table. The resolution of the lookup table refers to the amount of decimal points per pixel, a resolution of 100 implies that a pixel is divided into 100 points and therefore a pixel location can be specified in increments of 0.01 pixels.

The three Gaussian lookup table parameters as well as the star kernel size were all experimentally determined in section 4.1.1 by analysing which combination delivered the highest centroid generation accuracies with pixel intensities that is detectable by the image sensor on the star tracker, and a kernel size that will not result in unacceptable projection execution times.

3.1.4.2 Generating A Star Kernel

This section discusses how a star kernel is generated with the use of the Gaussian lookup table. To determine which intensity value must be used for each pixel, firstly the distance Δx and Δy that the star centroid lies from the middle of a pixel is calculated for the x- and y-axis respectively, as illustrated in figure 3.3. In figure 3.3, the point (C_x, C_y) is the centroid of a star and point (P_x, P_y) is the centre of the pixel.

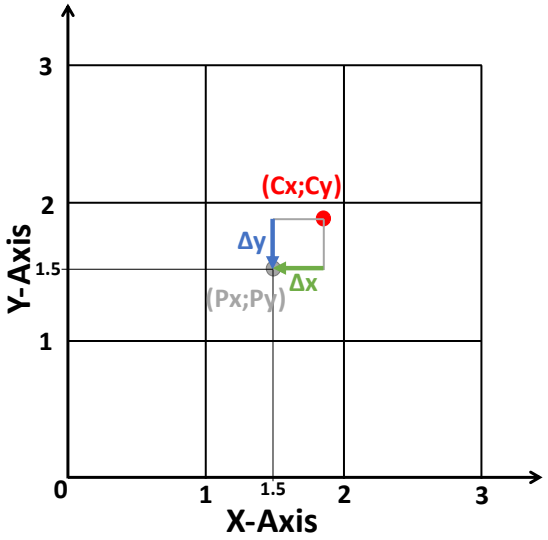


Figure 3.3: Determining Star Centroid Distance From Pixel Centroid

For this project the centre of a pixel is defined at an offset of 0.5 from the edges of the pixel, as illustrated with point (P_x, P_y) in figure 3.3. For this pixel, the centre lies at $(P_x, P_y) = (1.5, 1.5)$. The point $(1.0, 1.0)$ will be the bottom left corner and the point $(1.99, 1.99)$ will be the top right corner, the point $(2.0, 2.0)$ will therefore mark the start of a new pixel.

The Gaussian distributed lookup table values range from a minimum of 0 to a peak value of 255, this is the range of greyscale values where 0 represent a black pixel and 255 represent a white pixel. For each pixel in the star kernel, the values Δx and Δy are used to find the intensity on both the x- and y-axis respectively using the same lookup table for both axes. This is illustrated in figure 3.4 for the x-axis and a kernel size of 5x5 pixels. The total intensity of the current pixel is then calculated with equation 3.16.

$$P_i = \sqrt{I_x * I_y} \tag{3.16}$$

where

P_i: resulting pixel intensity
I_x: pixel intensity on the x-axis
I_y: pixel intensity on the y-axis

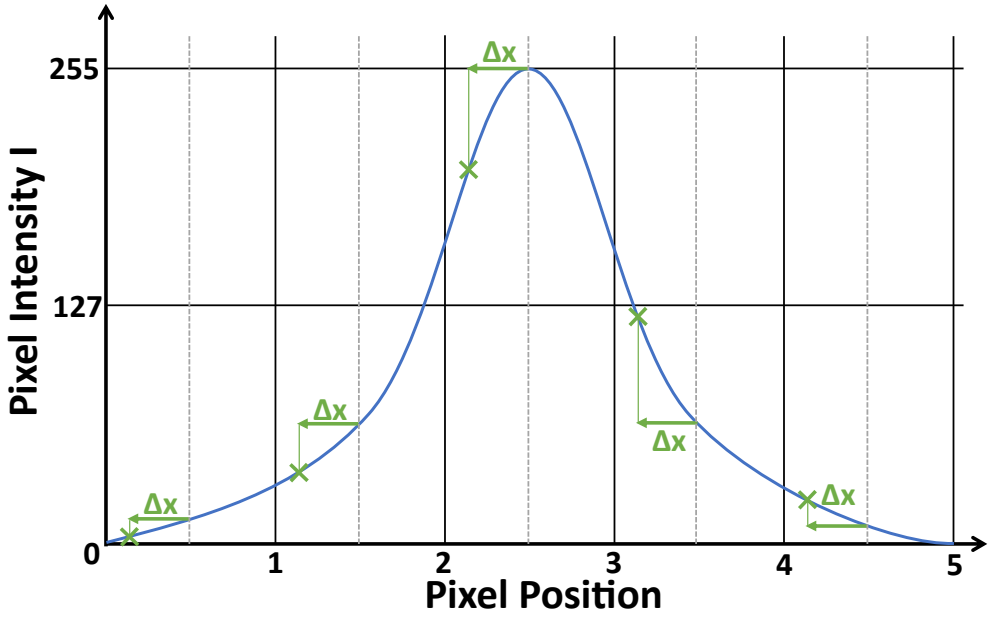


Figure 3.4: Finding Pixel Intensity Value On Gaussian Curve For The X-Axis

The pseudocode to calculate the star kernel for each star is shown in Algorithm 3. The input variables are the star centroid's x- and y-axis value c_x and c_y, an array containing the values of the Gaussian lookup table *GaussianLookup*, the resolution to which the lookup table was generated *LookupTableResolution* and lastly the dimension of the star kernel.

Algorithm 3: Generate Star Kernel

input : star centroid (cx, cy), $GaussianLookup$, $LookupTableResolution$, $KernelDimension$
output: $starKernel$

calculate decimal fraction value of cx as $decX = cx - \texttt{Floor}(cx)$
calculate decimal fraction value of cy as $decY = cy - \texttt{Floor}(cy)$
calculate x-axis offset from centre of pixel as $deltaX = (decX - 0.5) * LookupTableResolution$
calculate y-axis offset from centre of pixel as $deltaY = (decY - 0.5) * LookupTableResolution$
for $j = 0$ *to* $j = (KernelDimension - 1)$ **do**
    for $k = 0$ *to* $k = (KernelDimension - 1)$ **do**
        pixel intensity value for x-axis: $pixIntensityX = GaussianLookup((k + 0.5) * LookupTableResolution - deltaX)$
        pixel intensity value for y-axis: $pixIntensityY = GaussianLookup((j + 0.5) * LookupTableResolution - deltaY)$
        $starKernel(j, k) = \texttt{Sqrt}(pixIntensityX * pixIntensityY)$
    end
end
return $starKernel$

3.2 Star Detection and Identification

Star detection and identification techniques are used to analyse images of stars, whether the image was generated by the star projection software or captured by a star tracker. The algorithms and techniques discussed in this section are implemented in a Matlab script and used for two purposes. Firstly, they are used in section 4.2 to determine the accuracy to which star centroids can be generated with the star projection software and the accuracy to which generated images of star constellations represent the corresponding requested attitudes. Secondly, they are used throughout chapters 5 and 6 to analyse actual star images captured by the CubeStar inside the emulation environment. The star detecting algorithms are used to find the star centroids for captured images of both calibration patterns and star constellations that are projected, and the star identification algorithms are used for captured images of star constellations that are projected.

The first step in processing an image is to identify what pixels in the image belongs to each respective star, and what pixels can be considered as noise or dead pixels. The techniques used to detect the stars consist of two algorithms, an Image Plane Search algorithm and a Region Growing Algorithm (RGA). Once all the stars are detected, the coordinates can then be transformed from image plane coordinates to sensor body coordinate vectors, these vectors are then used to find the identity of each star and eventually to estimate the attitude based on

the inertial vectors of the observed stars.

3.2.1 Image Plane Search

The Image Plane Search algorithm scans through the image to detect possible stars. Since the optics of star trackers are defocussed, it is known that stars on an image will occupy an area of pixels and not only a single pixel. The time it takes to detect stars can be drastically reduced by not searching each pixel, while still ensuring that each star is detected. Instead, only every third column and row is searched for a pixel, as illustrated in figure 3.5.

Just as images of real stars, images taken of projected stars will always have noise present. For real stars, the noise sources can come from electronics on the camera or satellite, or in the case of ground testing the noise can come from light pollution from cities and the sun's reflection from the moon. When projecting stars on a monitor the main sources of noise come from the star tracker's electronics, dead pixels on the image sensor and the backlight of the monitor on which the stars are projected. To distinguish between noise and pixels that are part of a star, an intensity threshold must be determined so that when a checked pixel's intensity falls above this threshold it invokes the RGA. This intensity threshold is used to distinguish between pixels that belong to a star and pixels that can be considered as noise, the value of this threshold is experimentally determined in section 4.1.1.

3.2.2 Region Growing Algorithm

The RGA is a recursive algorithm that finds all the pixels belonging to a single star. Given a starting pixel (referred to as the "seed" pixel), the RGA checks all four neighbour pixels after storing the current seed's location and setting its intensity to zero [20]. If a neighbour pixel falls above the intensity threshold, the RGA is invoked again with the neighbour pixel as the seed. Figure 3.5 provides a visual representation of the image plane search and region growing algorithms.

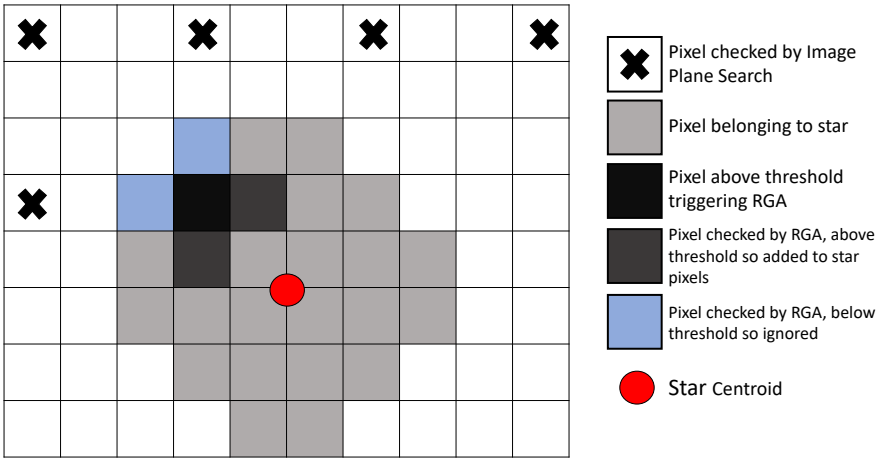


Figure 3.5: Image Plane Search and Region Growing Algorithm

3.2.3 Centroid Detection

After a star is detected and every pixel belonging to each respective star is stored, the centroid locations of the stars can be calculated. The Centroiding Algorithm used in this project is the same that is used by the CubeStar [6], it is a weighting algorithm used to calculate the centre of a star by weighing each pixel intensity with its x- and y-axis location. The equations used to calculate the centroid of a star in the two-dimensional sensor image plane reference frame is shown in equations 3.17 and 3.18 below [6].

$$centroid_x = \frac{\sum_{n=0}^{numpixels}(pixel[n]_x) * (pixel[n]_{intensity})}{\sum_{n=0}^{numpixels}(pixel[n]_{intensity})} \tag{3.17}$$

$$centroid_y = \frac{\sum_{n=0}^{numpixels}(pixel[n]_y) * (pixel[n]_{intensity})}{\sum_{n=0}^{numpixels}(pixel[n]_{intensity})} \tag{3.18}$$

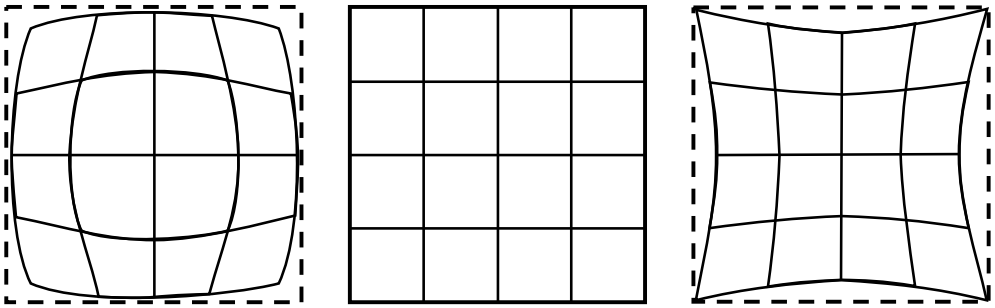
where:

$centroid_x$: location of star centroid on the x-axis in pixels
$centroid_y$: location of star centroid on the y-axis in pixels
$numpixels$: total number of pixels belonging to current star
$pixel_x$: location of pixel on the x-axis
$pixel_y$: location of pixel on the y-axis
$pixel_{intensity}$: pixel intensity in greyscale

3.2.4 Distortion Correction

For a camera that has no lens distortion present, if the standard pinhole camera model is used each point on the sensor image plane will correspond exactly to its real world location. This, however, is not the case since all cameras will have distortion present that cause large errors in observed points that needs to be corrected [21],[22]. The distortion can be caused by either misaligned hardware in the sensor element and the lens, or by imperfections in the optics of the lens [6],[23]. The most common forms of distortion are radial and tangential distortion.

Radial distortion is symmetric about the boresight and is divided into two types: barrel distortion and pincushion distortion. The two types are shown in figure 3.6, barrel distortion results in points appearing further from the boresight than expected, while pincushion distortion results in points appearing closer to the boresight than expected. The effect of both barrel and pincushion distortion increases further away from the boresight, so the points that are furthest from the boresight will have much larger errors than the points that are closer to the boresight.



(a) Image With Barrel Distortion

(b) Image With No Distortion

(c) Image With Pincushion Distortion

Figure 3.6: Lens Distortion

There are various distortion models from other work that can be used to undistort image points once the calibration values for the model is obtained. Duane Brown first proposed the decentering distortion model in 1966 [24] that can correct both radial and tangential distortion. However, the calibration results in Erlank's work [6] showed that the tangential component of the CubeStar's distortion is so minimal that it can be ignored without a significant decrease in distortion correction accuracy. Therefore, CubeSpace has opted for a distortion model that only compensates for radial distortion [23] and the decision was repeated for this project as it would simplify the calibration process significantly if only radial distortion coefficients has to be estimated.

The distortion model used in this project is an even-order polynomial radial distortion model adapted from [21], the distortion model is described by equations 3.19 and 3.20.

$$x_u - x_o = x_t(1 + K_1 r_d^2 + K_2 r_d^4 + K_2 r_d^6) \tag{3.19}$$

$$y_u - y_o = y_t(1 + K_4 r_d^2 + K_5 r_d^4 + K_6 r_d^6) \tag{3.20}$$

with

$$x_t = (x_d - x_o) \quad \text{and} \quad y_t = (y_d - y_o) \tag{3.21}$$

and

$$r_d = \sqrt{(x_d - x_o)^2 + (y_d - y_o)^2} \tag{3.22}$$

where

x_u, y_u: undistorted point in pixels
x_d, y_d: distorted point in pixels
x_o, y_o: centre of distortion
$K_1 \ldots K_6$: radial distortion coefficients

From these equations, the values that need to be solved for are the distortion coefficients $K1$ to $K6$ and the centre of distortion (x_o, y_o), a calibration method to solve for these values is discussed in section 5.2.3.

3.2.5 Image Plane to Unit Vector

The star centroid locations must now be converted from the two-dimensional sensor image plane reference frame to the three-dimensional sensor-body reference frame. As discussed in section 3.1.3, to project a three dimensional point onto a two-dimensional image plane is fairly simple when using the pinhole camera model. The reverse of this problem is more complex but can be achieved with equation 3.23 that is adapted from Erlank [6], who has an extensive derivation of this equation. The notation of the equation relates to the pinhole camera model in figure 3.2, it is used to transform a point on the sensor plane (u, v) to the sensor-body coordinate system (x, y, z).

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (u_c - u_o)\frac{p_x}{f}[1 + ((u_c - u_o)\frac{p_x}{f})^2 + ((v_c - v_o)\frac{p_y}{f})^2]^{-\frac{1}{2}} \\ (v_c - v_o)\frac{p_y}{f}[1 + ((u_c - u_o)\frac{p_x}{f})^2 + ((v_c - v_o)\frac{p_y}{f})^2]^{-\frac{1}{2}} \\ [1 + ((u_c - u_o)\frac{p_x}{f})^2 + ((v_c - v_o)\frac{p_y}{f})^2]^{-\frac{1}{2}} \end{bmatrix} \tag{3.23}$$

where

x, y, z: components of unit vector in sensor-body coordinates
u_c, v_c: star centroid location in pixels
u_o, v_o: image plane origin in pixels
p_x, p_y: pixel pitches of the sensor in mm
f: focal length of the lens in mm

3.2.6 Star Matching

After all of the stars have been detected and their sensor-body coordinates are determined, the next step is to match the stars to their counterpart in the star catalogue. Once this is achieved, the matching vectors for each star in ECI coordinates can be found from the star vector list to enable attitude estimation. Although there are several star matching algorithms that are effective, the one used in this project is the Geometric Voting Algorithm (GVA) as this is the same that is onboard the CubeStar and proven to be of sufficient accuracy [6].

3.2.6.1 Star Distance List

The first requirement for the star matching process is a list containing the stellar distances between all the visible star pairs. The list is generated by calculating the angular distance between each star pair in the Star Vector List discussed in section 3.1.1 and adding it to the star distance list if it falls within a maximum angular distance. The maximum angular distance is set to the horizontal FoV of the CubeStar's lens, since both stars of a star pair that has an angular distance larger than the FoV of the lens will not be present on the image plane at the same time. Equation 3.24 is used to calculate the angular distance in radians of a star pair, where x, y and z are the ECI unit vector components of each star that is extracted from the Star Vector List.

$$D_{jk} = \cos^{-1}(x_j x_k + y_j y_k + z_j z_k) \tag{3.24}$$

An extract of this list is given in table 3.2, the list is sorted in increasing order for angular distances. Each entry only contains the HIP identification number for each star and their angular distance in radians. From the generated angular distance list, three pairs of stars with very small angular distances (as seen in the first three entries in table 3.2) proposed possible problems that needed to be addressed.

The first two pairs of stars, with angular distances 0.000075 rad and 0.001681 rad respectively, will lie approximately 0.047 and 1.874 pixels apart respectively. The third pair of stars with an angular distance of 0,006745 rad will lie approximately 7.51 pixels apart. Therefore, the first two pairs of stars that lie less than two pixels apart will definitely intersect each other when they are projected, while the third pair of stars will not intersect as long as the star kernel size that is generated does not exceed 7x7 pixels.

Table 3.2: Star Distance List

HIP_1	HIP_2	Angular Distance (rad)
71681	71683	0.000075
82514	82545	0.001681
17702	17847	0.006744
⋮	⋮	⋮
66657	90422	0.785386

Since the projection software can control which stars are projected, it was decided not to include stars 71683 and 82545 to avoid the issue of two stars overlapping. In section 4.1.1 it is decided to use a 7x7 kernel for all the stars, therefore both stars of the third pair could remain in the list to be projected.

3.2.6.2 Geometric Voting Algorithm

The GVA, first proposed in 2008 by Kolomenkin [25], is used to match stars detected on the image plane to stars in a star catalogue with the use of their sensor-body vectors and their matching ECI vectors from the catalogue. The GVA can be used during a Lost-In-Space (LIS) phase of a satellite, which means that it can estimate the attitude of the spacecraft without any prior knowledge of its attitude.

Each imaged star, S, is assigned its own list of votes, V. The angular distances D_{jk} (referred to as the distances) of each imaged star pair S_j and S_k are calculated. A region R_{jk} around the imaged distance is defined such that:

$$R_{jk} = [D_{jk} - e_{jk}, D_{jk} + e_{jk}] \tag{3.25}$$

where e_{jk} is an added error factor to take into account the possible inaccuracies in the centroid calculation caused by the detection noise interference of the star tracker's image sensor. For generated images, this error factor should be set very low as there are no detection noise present. For captured images, the error factor can be determined iteratively to find the value that delivers the most successfully

identified stars. Each catalogue distance is checked whether it lies within the region R_{jk} and if so, the IDs for each catalogue pair of stars are assigned to each of the imaged star pair's voting lists V_j and V_k respectively. For example, if the catalogue angular distance between catalogue stars 17702 and 17847 falls in the region R_{jk}, then both star IDs 17702 and 17847 are added to each voting list V_j and V_k. With the error factor e_{jk} multiple catalogue star pair distances will lie in this region for each imaged star pair, therefore imaged stars will have multiple possible catalogue star IDs in their voting list. This step is referred to as the first round of voting, the round is complete when all imaged star pairs were considered.

More often than not, the catalogue star that appear most often in an imaged star's voting list is the true identity of that imaged star, therefore the star that appears most often in an imaged star's voting list is then assigned as the imaged star's true identity, St_j. This process, however, requires a verification stage known as the second round of voting, since there can still be falsely assigned stars.

The second round of voting verifies that the distance $|St_j - St_k|$ of each matched pair of stars still lies within the region R_{jk} of their catalogue star counterparts. If so, each star in the pair receives a verification vote. At the end of this voting round, each star will have either many or close to no votes. The stars who have close to the maximum number of verification votes are considered to be correctly assigned, while stars that received less than a certain threshold are considered falsely identified.

The GVA can further be used in an "Assisted Matching" phase of the satellite (as opposed to the LIS phase), this is when the satellite receives rough attitude estimates from its many other attitude estimating sensors. In this phase, the star tracker is supplied with an estimate of the boresight-pointing vector and a reduced star catalogue can be generated, increasing the speed of the star matching process drastically. However, since this project will only make use of the star tracker as an attitude sensor and the GVA will be executed on a computer with a much faster processing speed than that of star trackers, this step was excluded.

3.3 Attitude Estimation

Attitude estimation is the process of finding the rotation matrix $\boldsymbol{R}_I^B$ that describe the rotation between the inertial reference frame and the sensor-body reference frame, and thus describes the attitude of the spacecraft. Attitude estimation algorithms use pairs of vectors in both the inertial reference frame (referred to as the reference vectors $\boldsymbol{v}_i$) and the sensor-body reference frame (referred to as the measured vectors $\boldsymbol{v}_b$).

The TRIAD estimator discussed in section 3.1.2 is one of the oldest methods for estimating the attitude of a spacecraft. While this method is very effective for the initial attitude calculation required to project the stars onto the monitor, it only uses two vector pairs while star trackers provide much more. Other solutions

exist that uses multiple vector pairs to provide more accurate attitude estimates, these solutions include Wahba's Problem, Davenport's Q-Method and the QUEST algorithm that are all discussed in this section.

3.3.1 Wahba's Problem

Without measurement errors, $\boldsymbol{v}_{kb} = \boldsymbol{R}_I^B \boldsymbol{v}_{ki}$ (where k refers to the vector pair) will be true for each vector measurement. Since there are always measurement errors present, this is never the case. Wahba first described the problem of finding the best fit for $\boldsymbol{R}_I^B$ over all vector pairs by minimising the cost function [26], [27]:

$$J(\boldsymbol{R}_I^B) \equiv \frac{1}{2}\sum_{k=1}^{N} \omega_k |\boldsymbol{v}_{kb} - \boldsymbol{R}_I^B \boldsymbol{v}_{ki}|^2 \tag{3.26}$$

where
J: cost function to be minimized
N: total number of vector measurements
ω_k: weight assigned to vector pair
The loss function can be expanded and rewritten as

$$J(\boldsymbol{R}_I^B) = \lambda_o - trace(\boldsymbol{R}_I^B \mathbf{B}^T) \tag{3.27}$$

where

$$\lambda_o = \sum_{k=1}^{N} \omega_k \quad and \quad \boldsymbol{B} \equiv \sum_{k=1}^{N} \omega_k (\boldsymbol{v}_{kb} \boldsymbol{v}_{ki}^T) \tag{3.28}$$

Various solutions to Wahba's problem exist [28], such as Singular Value Decomposition, Davenport's Q-Method and the QUEST algorithm. This project used the QUEST method, therefore only Davenport's Q-Method and the QUEST method is discussed.

3.3.2 Davenport's Q-Method

Davenport's Q-Method, by Paul Davenport [28], was the first proper solution to apply Wahba's problem to the attitude determination of a spacecraft. The attitude matrix $\boldsymbol{R}_I^B$ is parameterised by using a quaternion

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_v \\ q_4 \end{bmatrix} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4 \tag{3.29}$$

such that

$$\boldsymbol{R}_I^B = (q_4^2 - |\boldsymbol{q}_v|^2)\boldsymbol{I}_{3\times 3} + 2\boldsymbol{q}_v\boldsymbol{q}_v^T - 2q_4[\boldsymbol{q}_v]_\times \tag{3.30}$$

where $\boldsymbol{I}_{3\times 3}$ is a 3x3 identity matrix. From this attitude representation's properties, the trace in equation 3.27 can be expanded such that

$$trace(\boldsymbol{R}_I^B \mathbf{B}^T) = \boldsymbol{q}^T \boldsymbol{K} \boldsymbol{q} \tag{3.31}$$

where $\boldsymbol{K}$ is defined as

$$\boldsymbol{K} \equiv \begin{bmatrix} \boldsymbol{S} - \boldsymbol{I}s & \boldsymbol{z} \\ \boldsymbol{z} & s \end{bmatrix} \tag{3.32}$$

with

$$\boldsymbol{S} \equiv \boldsymbol{B} + \boldsymbol{B}^T \tag{3.33}$$

$$\boldsymbol{z} \equiv \begin{bmatrix} B_{23} - B_{32} \\ B_{31} - B_{13} \\ B_{12} - B_{21} \end{bmatrix} = \sum_k \omega_k \boldsymbol{v}_{kb} \times \boldsymbol{v}_{ki} \tag{3.34}$$

$$s = trace(\boldsymbol{B}) \tag{3.35}$$

The optimal quaternion that solves the optimisation problem is determined by finding the normalized eigenvector of $\boldsymbol{K}$ with the largest eigenvalue (λ_{max}) such that

$$\boldsymbol{K}\boldsymbol{q}_{opt} \equiv \lambda_{max}\boldsymbol{q}_{opt} \tag{3.36}$$

To solve for the optimal quaternion, the QUEST method provides a more efficient solution to the eigenvalue problem while claiming similar accuracies.

3.3.3 QUEST Algorithm

The QUEST algorithm was developed by Malcolm Shuster [29] to approximate the Q-Method for decreased computationally expensive processes that the eigenvalue of the Q-Method involves. The full derivation of the QUEST algorithm can be found in [29], the algorithm approximates the optimal eigenvalue λ_{opt} as

$$\lambda_{opt} \approx \lambda_o \tag{3.37}$$

The attitude quaternion $\bar{\boldsymbol{q}}$ can be found by first solving the Rodrigues Parameters $\boldsymbol{p}$ in equation 3.38 by using Gaussian Elimination.

$$\boldsymbol{p} = [(\lambda_o + s)\boldsymbol{I}_{3\times 3} - \boldsymbol{S}]^{-1}\boldsymbol{z} \tag{3.38}$$

The attitude quaternion $\bar{\boldsymbol{q}}$ is then calculated with

$$\bar{\boldsymbol{q}} = \frac{1}{\sqrt{1 + \boldsymbol{p}^T\boldsymbol{p}}} \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} \tag{3.39}$$

where

$$\bar{\boldsymbol{q}} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4 \tag{3.40}$$

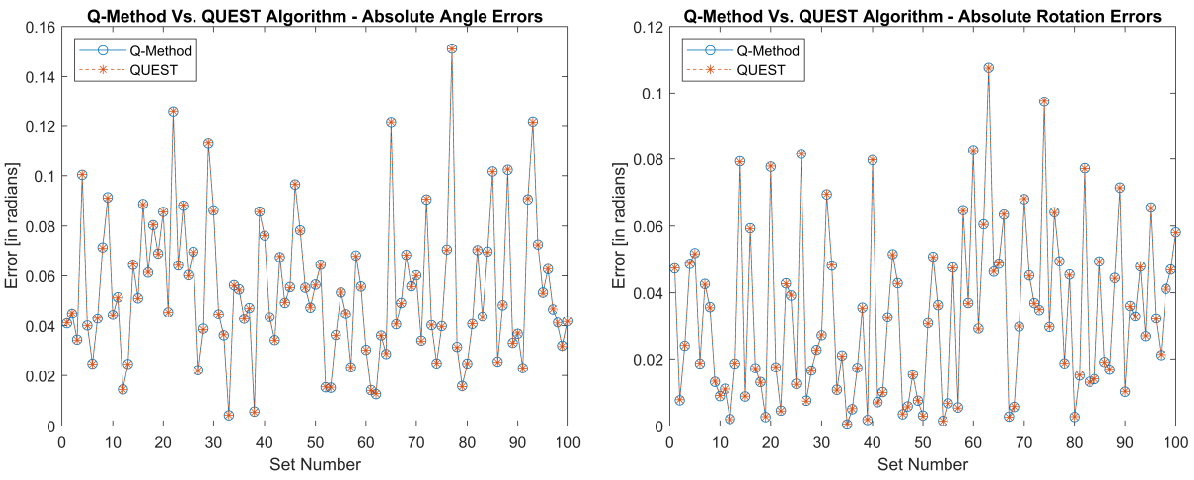
3.3.3.1 Davenport's Q-Method versus QUEST Algorithm

The accuracy and execution time of both methods were evaluated in a simulation where 20 random vector pairs with random noise induced were generated and used to estimate the attitude based on the generated vector pairs using both the Q-Method and the QUEST algorithm, the simulation was then repeated 100 times. To compare the accuracies of both methods the two estimated rotation matrices $R_{qmethod}$ and R_{quest} are compared to the generated rotation matrix R_{exact}. For each rotation matrix $R_{qmethod}$ and R_{quest}, the angle error from the estimated boresight pointing vector to the generated boresight pointing vector is calculated as well as the rotation error around this pointing vector (this technique is discussed in the next section).

The steps for the simulation that is repeated 100 times are as follows:

1. Generate rotation matrix R_{exact} from inertial reference frame to body reference frame.
2. Generate and normalize 20 random inertial unit vectors.
3. For each inertial vector, calculate body unit vector counterpart by multiplying inertial vector with rotation matrix R_{exact}.
4. Induce a random noise value in the interval $(-0.005, 0.005)$ to each vector component of each body vector. This results in a maximum absolute error of $\arctan(\frac{0.005}{1}) = 0.2865°$.
5. Normalize the body vectors.
6. Estimate attitude $R_{qmethod}$ from the 20 vector pairs using the Q-Method and log the execution time.
7. Estimate attitude R_{quest} from the 20 vector pairs using the QUEST algorithm and log the execution time.
8. Calculate angle error between the pointing vectors of $R_{qmethod}$ and R_{exact} as well as the rotation error.
9. Calculate angle error between the pointing vectors of R_{quest} and R_{exact} as well as the rotation error.

The results of one simulation are shown in figure 3.7. Figure 3.7a shows the absolute angle errors of both methods for each set (of the 100) and figure 3.7b shows the rotation errors. For this simulation the average angle errors resulted in 0.054934 radians for the Q-Method and 0.054919 radians for the QUEST algorithm, which is a difference of only 3.1 arcseconds. The average rotation errors resulted in 0.032910 radians for the Q-Method and 0.032890 radians for the QUEST algorithm, which is a difference of only 4.3 arcseconds.



(a) Angle Errors (b) Rotation Errors

Figure 3.7: Q-Method Versus QUEST Algorithm Attitude Estimation Errors

The execution times of both methods for each set in this simulation are shown in figure 3.8. The average execution time resulted in 63.83µs for the Q-Method and 31.46µs for the QUEST algorithm, which is a difference of 32.36µs and indicates that the QUEST algorithm is approximately 2.03 times faster than the Q-Method. The execution time spikes that are observed in figure 3.8 is likely due to the Central Processing Unit (CPU) of the computer executing multiple tasks simultaneously at those particular sets.

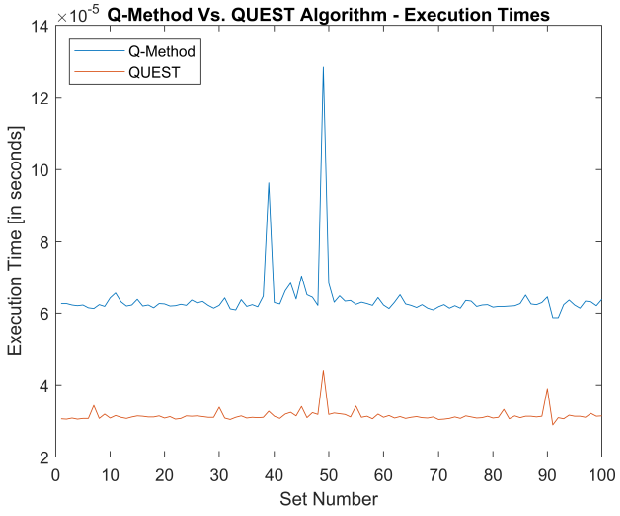


Figure 3.8: Q-Method Versus QUEST Algorithm Execution Times

Since the attitude estimation algorithms for this project will only be executed on a computer, the small execution times show that using either method above the other will not have any noticeable difference. Furthermore, the small differences in the angle- and rotation errors also show that using either method will not result in noticeable errors.

3.3.4 Attitude Error Calculation

To determine the accuracy to which attitudes are generated and estimated the error between two given attitudes must be calculated. To calculate the error between two attitude matrices $\boldsymbol{R_a}$ and $\boldsymbol{R_b}$, firstly the error rotation matrix $\boldsymbol{R_{error}}$ is calculated with equation 3.41 below. Note that for equation 3.41 to be used a rotation quaternion would first need to be converted to a rotation matrix with the use of equation 2.19.

$$\boldsymbol{R_{error}} = \boldsymbol{R_a^T R_b} \tag{3.41}$$

The error rotation matrix $\boldsymbol{R_{error}}$ is then converted to an error rotation quaternion $\boldsymbol{q_{error}}$ with equations 2.20. From the first three elements in the error rotation quaternion the unit vector $\boldsymbol{u_{error}}$ (from equation 2.18) is extracted by using equation 2.16 such that

$$\boldsymbol{u_{error}} = \frac{\boldsymbol{q_{error}}(1:3)}{\sin(\frac{\theta}{2})} \tag{3.42}$$

with

$$\boldsymbol{u_{error}} = \begin{bmatrix} u_{ex} & u_{ey} & u_{ez} \end{bmatrix}^T \tag{3.43}$$

The angle θ about the unit vector $\boldsymbol{u_{error}}$ is extracted from the fourth element of the error rotation quaternion with equation 2.17 such that

$$\theta = 2\arccos(\boldsymbol{q_{error}}(4)) \tag{3.44}$$

The total attitude error is then represented by an angle error and a rotation error. The angle error represents the angle between the two boresight vectors from attitude matrices $\boldsymbol{R_a}$ and $\boldsymbol{R_b}$, and the rotation error represents the difference between the rotation about the boresight of attitude matrix $\boldsymbol{R_a}$ and the rotation about the boresight of attitude matrix $\boldsymbol{R_b}$. The angle- and rotation errors are calculated with the following equations:

$$AngleError = \theta\sqrt{u_{ex}^2 + u_{ey}^2} \tag{3.45}$$

$$RotationError = \theta u_{ez} \tag{3.46}$$

With the equations discussed in this section the error between a generated and an estimated attitude can be calculated.

3.4 Chapter Summary

This chapter was divided into three parts, star projection, star detection and identification, and attitude estimation. The first part, star projection, discussed all the necessary algorithms and techniques required to project a star with an accurate profile to real night sky stars onto a monitor. The second part of this chapter, star detection and identification, discussed all the necessary algorithms required to detect stars that are present on a generated or captured image and identify them. The final section discusses three different methods for attitude estimation and provides an explanation on why the QUEST algorithm was chosen for this purpose, and lastly the method used to calculate the errors between two attitudes was explained.

Chapter 4

Software Design and Analysis

With all the required algorithms investigated to project stars and analyse star images, the star projection techniques and algorithms were implemented in a software program created to project stars onto a monitor to satisfy project objectives 1 and 6. This chapter provides an overview on the basic functioning of the software program in the first section, for the stars to be projected with the accurate star profiles discussed in section 3.1.4 the optimal values for the Gaussian lookup table parameters are also discussed in this section.

To ensure that the star projection software produces accurate star projections to satisfy project objective 1, the second section explains the method that was used to analyse the generated star images and the results are presented.

To satisfy project objective 6, the third section presents the star tracker kinematic equations that are used to update the star tracker's attitude that will simulate constant angular rates. To achieve accurate angular rate projections, a reliable software timer is investigated.

4.1 Software Overview

The star projection software for this project was written in Microsoft's Visual Studio in the C♯ programming language, this allowed the use of the free and very powerful graphics library, Windows Forms, that Microsoft includes in the .NET Framework.

The stars are projected onto a Graphical User Interface (GUI) that can be displayed on the monitor to be captured by the star tracker. During software testing and analysis, the GUI contains a control panel with some controls for the program, and the coordinates of the current boresight pointing vector is displayed in various attitude representations. The identification number of each star on the image can also be displayed for easier navigation when searching for specific stars. At start-up, a default GUI is generated where the star "Alpha Crux" from the well known constellation, the Southern Cross, is set as the boresight pointing star. The boresight pointing star can be changed to any star that is in the star catalogue

and the corresponding attitude matrix will be calculated by using the TRIAD estimator discussed in section 3.1.2. The default GUI can be viewed in figure A.1 in appendix A. During emulation all features are switched off so that only the projected stars are present on the monitor.

Once the stars are projected, certain keys on a keyboard can be pressed to move the boresight of the star tracker to a different location. Each key press results in a 1° increment or decrement of the boresight angle by setting the corresponding Euler angle in equation 2.9 to 1° and performing a 2-1-3 Euler Angle rotation with equations 2.10 to 2.12 (discussed in section 2.4.2). The key presses and their resulting rotations are summarized in table 4.1 below.

Table 4.1: Key Presses On Keyboard To Move Star Tracker Boresight

Key Pressed	Euler Angle Value	Axis Of Rotation	Resulting Star Movement
'W'	$\theta = 1°$	Positive rotation about X-axis	Bottom to top
'S'	$\theta = -1°$	Negative rotation about X-axis	Top to bottom
'A'	$\phi = -1°$	Negative rotation about Y-axis	Right to left
'D'	$\phi = 1°$	Positive rotation about Y-axis	Left to right
'Q'	$\psi = 1°$	Positive rotation about Z-axis	Counter-clockwise rotation
'E'	$\psi = -1°$	Negative rotation about Z-axis	Clockwise rotation

4.1.1 Star Profile Determination

For the stars to be projected, the three Gaussian lookup table parameters discussed in section 3.1.4 as well as the star kernel size must first be determined. The parameters were determined experimentally by generating star kernels with varying parameters and evaluating their accuracy, as will be discussed in this section.

To analyse the results of the experiment a pixel intensity threshold value for the RGA, as mentioned in section 3.2.2, is required. This threshold value is used to see how many pixels in the star kernel will be generated below the CubeStar's sensor sensitivity. Since generated pixels below this intensity will most likely not be detected by the CubeStar, the accuracy provided by these pixels will not be obtained and therefore these pixels must also be excluded when analysing the results of this experiment. The final threshold, $IntensityThreshold$, was determined by analysing the noise levels in images taken by the CubeStar in the emulation environment once it was built (discussed in chapter 5). An intensity threshold of 5 was chosen for $IntensityThreshold$.

Since the centroid of a star can be located at any location within a single pixel, for the experiment a star kernel was generated at each x- and y-axis location within a pixel to a position resolution of 0.01 pixels, resulting in 10000 kernels generated at different centroid locations. Each kernel's generation error was then calculated

by finding its measured centroid (as calculated by the centroiding algorithm with the *IntensityThreshold* in place) and comparing it to the generated centroid. This process was repeated with varying Gaussian lookup table parameter values.

The results for the different sets of star profile values are summarized below in table 4.2, a 5x5 kernel size is not included in the summarized table as the only set of parameters that yielded better accuracies than bigger kernel sizes delivered pixel intensities below the *IntensityThreshold* for 16% of the total pixels, therefore this kernel size was not considered. However, the full table can be seen in detail in table B.1 in appendix B. It was observed that a resolution higher than 1000 (or 0.001 pixels) would not yield lower errors irrespective of the standard deviation or kernel size. It was also observed that for a star kernel size of 9x9, errors that proved better than that of a smaller kernel size delivered a large percentage of pixel intensities below the *IntensityThreshold*. For a star kernel size of 9x9, at best 24.69% of the total pixels are below the threshold and will not be detected by the CubeStar and therefore the accuracy provided by this larger kernel size will not be obtained.

Table 4.2: Summarized Gaussian Lookup Table Parameters Accuracies

	Kernel Size:	7x7		9x9	
Resolution	Standard Deviation	Maximum Error [pixels]	% Pixels Below Threshold	Maximum Error [pixels]	% Pixels Below Threshold
10	0.9	0.0640	24.49	0.0514	54.3210
10	1.0	0.0769	8.16	0.0544	44.4444
10	1.1	0.0979	0.00	0.0600	24.6914
100	0.9	0.0252	24.49	0.0027	54.3210
100	1.0	0.0470	8.16	0.0081	44.4444
100	1.1	0.0736	0.00	0.0177	24.6914
1000	0.9	0.0252	24.49	0.0027	54.3210
1000	1.0	0.0470	8.16	0.0081	44.4444
1000	1.1	0.0736	0.00	0.0177	24.6914

The star profile variables that were chosen are summarized in table 4.3. This set of variables delivered a high accuracy while maintaining a close representation of a real star, and with a percentage of pixel intensities below *IntensityThreshold* of 8.16% only the 4 pixels with the lowest intensities of the 49 might not be detected by the CubeStar.

Table 4.3: Star Kernel Variable Values

Variable Name	Value
Kernel Size	7x7 pixels
Standard Deviation	1.0
Lookup Table Resolution	100

The values in the above tables produced generated stars that is shown in figure 4.1 below. The figure shows a star with a centroid that lies in the middle of a pixel (figure 4.1a) as well as a star with a centroid that lies in the corner of a pixel (figure 4.1b). This shows the effect of using the Gaussian star profile and shifting the pixel intensities along a Gaussian lookup table with the method described in section 3.1.4, and as shown in table 4.2 the generation errors will not exceed 0.0470 pixels on any axis.

(a) Centroid In Centre Of Pixel (b) Centroid In Corner Of Pixel

Figure 4.1: 7x7 Generated Star Kernel

4.2 Generated Star Image Analysis

This section discusses the analysis performed to determine the position projection accuracy of the generated star images. To evaluate the performance of the star tracker in the emulation environment it is important to know the position projection accuracy to which star images are generated and projected.

With the star projection software, twenty randomly selected boresight pointing stars were used to generate star images at their requested attitude. The generated images were analysed by implementing the star detection algorithms discussed in section 3.2 in a Matlab script. Figure 4.2 below shows the generated centroid errors in the x- and y-axis. To determine these errors, for each star the calculated centroid from the catalogue where the star is generated on the image is compared to the resulting generated centroid calculated with the star detection algorithms. The average centroid errors between the 20 sets resulted in 0.0181 pixels in the x-axis and 0.0185 pixels in the y-axis.

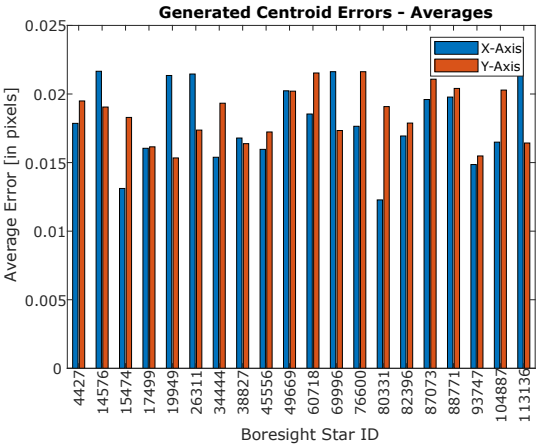


Figure 4.2: Generated Centroid Errors

The attitude generation was also analysed to investigate what effect the centroid errors mentioned above have on the generated attitudes. As discussed in section 3.3.4 an attitude error is represented with two values: the angle error which is the angle between two boresight unit vectors and the rotation error which is the difference between the rotation about the first boresight unit vector and the rotation about the second boresight unit vector. To find the angle- and rotation errors for the twenty generated images equations 3.41 to 3.46 are implemented on the requested attitude and the resulting generated attitude estimated by the QUEST algorithm for each image. Figure 4.3 below shows the generated attitude errors in arcseconds for each boresight pointing star image. The average errors between the 20 sets resulted in a 2.47 arcseconds average angle error and a 1.45 arcseconds average rotation error.

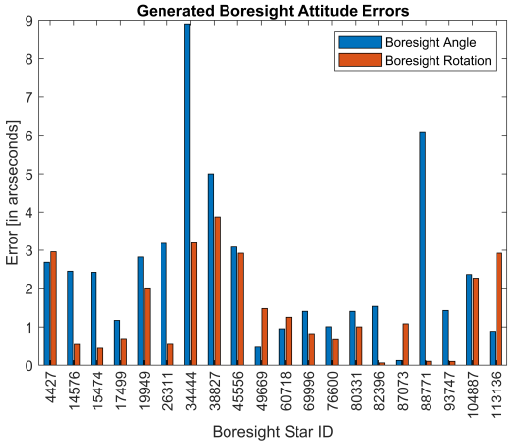


Figure 4.3: Generated Attitude Errors

The results discussed in this section shows that star images are projected with a very high accuracy, this will ensure the validity of the performance analysis of images captured by a star tracker in the emulation environment.

4.3 Constant Angular Rate Projection

This section explains how constant angular rates are achieved by moving the projected stars as well as the accuracy to which the rates could be simulated. The simulation accuracy must be determined to be able to verify the performance of the star tracker's tracking algorithm, if the stars are not projected with an accurate simulation the star tracker will also show poor performance. To simulate constant angular rates for a star tracker the projected stars can be moved appropriately over the monitor by updating the attitude of the star tracker with the kinematic equations that are presented in this section. To update the stars at accurate intervals a software timer is then investigated.

4.3.1 Star Tracker Kinematics

The kinematic equation for the movement of a satellite based on its angular rate is used to update the satellite's rotation matrix at discrete time steps depending on the sampling period that is specified by a timer's refresh rate. Firstly, the inertially referenced angular rates are specified in the angular rate vector ω such that

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_{ix} \\ \omega_{iy} \\ \omega_{iz} \end{bmatrix} \tag{4.1}$$

where

$\omega_{ix}, \omega_{iy}, \omega_{iz}$: inertially referenced angular rate about each axis in radians/second

To update the rotation matrix at discrete time steps, the kinematic equation expressed in 4.2 [15],[30] is applied.

$$\boldsymbol{R}_I^B(k+1) = - \begin{bmatrix} 0 & -w_{iz}T & w_{iy}T \\ w_{iz}T & 0 & -w_{ix}T \\ -w_{iy}T & w_{ix}T & 0 \end{bmatrix} \boldsymbol{R}_I^B(k) \tag{4.2}$$

where

$\boldsymbol{R}_I^B$: rotation matrix describing rotation from inertial to body reference frame
k: discrete time sample
T: sampling period in seconds

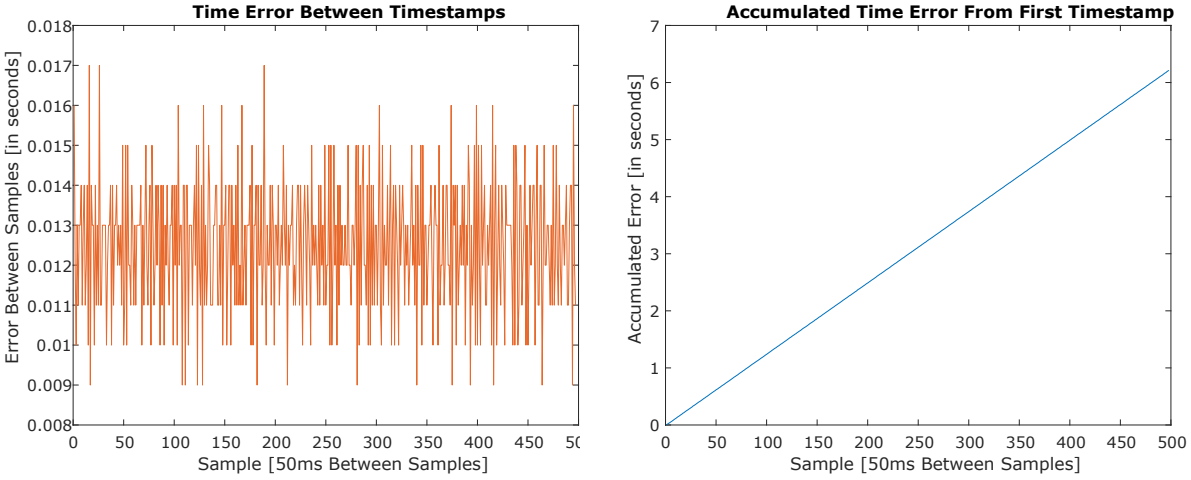
Each time the rotation matrix is updated, the stars that must be projected are also updated with their new centroid locations and therefore the stars move across the monitor, creating the simulated movement of the star tracker with the specified angular rates.

4.3.2 Software Timer

A software timer is used to trigger an update of the projected star positions at a certain refresh rate, therefore the accuracy of the simulated angular rates de-

pend entirely on the accuracy of the timer that is used. The timer will be used to implement the star tracker kinematic equations to find the updated attitude at a certain frequency, if the timer is unreliable the elapsed time will not be measured accurately and the stars could be updated with varying elapsed times between updates/time steps, this will cause the star tracker to either capture an image multiple times or miss the capture of an image. If an image is captured multiple times or not captured at all, the incorrect attitude will be represented for that time step which will cause the star tracker to calculate the wrong rate estimation, this in turn will produce inaccurate rate estimation by the star tracker due to inaccurate angular rate projections.

Upon investigation the default "Timer" class in Windows Forms showed very poor accuracy with high refresh frequencies. The accuracy was determined with different frequencies by documenting the timestamps at each sample tick of the timer over a 25 second period. For each sample the elapsed time from the previous sample is measured and compared to the refresh rate to determine the time error between samples, the elapsed time from the first sample to the current sample is also measured and compared to the expected elapsed time at the current sample to deliver an accumulated time error. For a 20Hz timer refresh rate the expected time between samples is 50ms, figure 4.4 below shows the results of the error analysis of the timestamps.

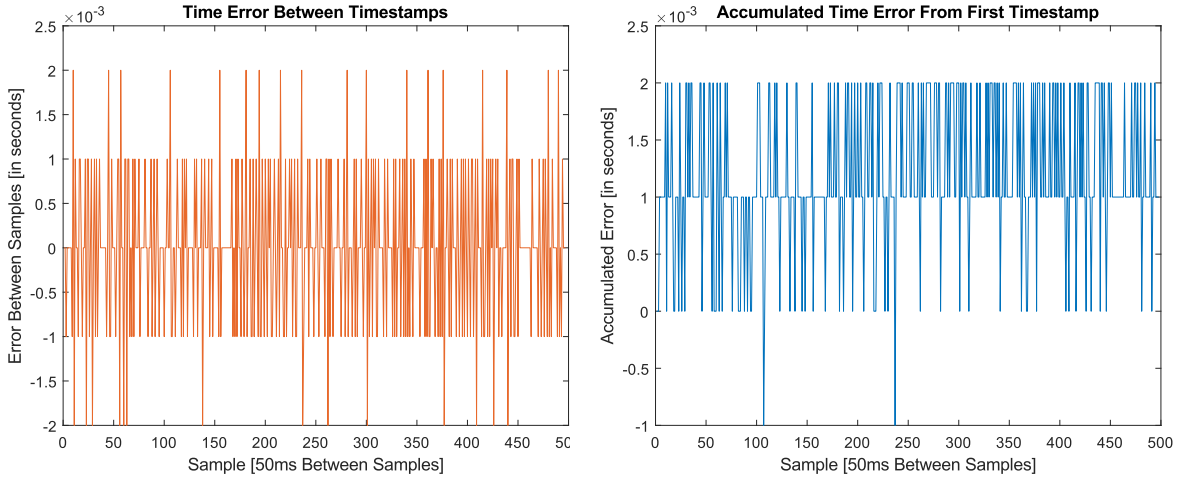


(a) Time Error Between Samples (b) Accumulated Time Error

Figure 4.4: 20Hz Windows Forms Timer Errors

The average error between the samples from figure 4.4a resulted in 12.5ms, figure 4.4b show that these errors resulted in an accumulated time error of 6.213s over a 25 second period and would increase further over time. This will cause large angular rate errors as the update frequency of the star projection will become completely desynchronized with the image frequency of the star tracker. This problem was solved by writing a custom timer class, "Accurate Timer", inspired by [31] with a much higher accuracy as it uses a "multimedia timer" from the

Multimedia Application Programming Interface (API) included in the Microsoft Windows operating system. The "Accurate Timer" class was evaluated similar to the Windows Forms timer, and the results are shown in figure 4.5 below.



(a) Time Error Between Samples (b) Accumulated Time Error

Figure 4.5: 20Hz Multimedia Timer Errors

From figure 4.5a, the multimedia timer showed an average error close to zero. This zero average error ensures that there is no accumulated time error, which is observed in figure 4.5b that shows no linear increase of the accumulated time error and instead a constant average error close to zero over a 25 second time period. This accurate timer will ensure that the star projection with constant angular rates will stay synchronised with the imaging frequency of the star tracker to deliver high simulation accuracy.

4.4 Chapter Summary

The purpose of this chapter was to give an overview of how the star projection software functions and to demonstrate that the star projection algorithms from section 3.1 could be implemented to produce highly accurate star images to be projected for a star tracker to capture, and therefore project objective 1 was satisfied. The first section gave an overview of the software and how it can be controlled, this section also discussed the determination of the values for the Gaussian lookup table parameters to produce accurate star profiles.

The second section discussed the analysis performed to determine the accuracy to which stars could be generated with the star projection software by using the star detection and identification algorithms discussed in section 3.2. The results showed that from twenty generated star images the average centroid errors resulted in 0.0181 pixels in the x-axis and 0.0185 pixels in the y-axis. The accuracy to which the twenty requested attitudes could be generated was also discussed in this section, the angle- and rotation errors of the generated images were calculated

by comparing the estimated attitude to the requested attitude with the method discussed in section 3.3.4. For the twenty generated star images the average angle error resulted in 2.47 arcseconds and the average rotation error resulted in 1.45 arcseconds.

The final section discussed the method used to move the projected stars across the monitor with constant angular rates to satisfy project objective 6. Firstly, the star tracker kinematic equations were presented, these equations are used to update the star tracker's attitude. To produce accurate constant angular rates, a software timer was presented that measures the elapsed time to trigger attitude updates at the correct intervals. The software timer that was created showed an average error of close to zero at a maximum refresh rate of 20Hz. This high accuracy allows the timer to check the elapsed time with a consistent frequency of 20Hz or slower, which in turn enables the stars to be reliably updated at the correct time to produce accurate constant angular rates.

Chapter 5

Emulation Environment Design and Analysis

With functioning star projection software that can project accurate star images, this chapter discusses the design and analysis of the emulation environment that is required to satisfy project objectives 2, 3, 4 and 5. In the first section of this chapter the physical design aspects of the emulation environment is discussed, along with the components that were chosen, designed and built. The distance that the CubeStar must be placed from the monitor is then investigated, followed by the discussion on the use of a secondary lens to adjust the focus of the CubeStar. The final emulation environment is shown at the end of this section.

Before the final emulation performance measurements of the CubeStar in the emulation environment can be performed, the first practical data is analysed to determine the effectiveness of the emulation environment. Firstly, an alignment correction method and its results are presented. Due to the presence of radial distortion in the lens of the CubeStar, a calibration method is presented that will remove the distortion errors from captured images. The calibration of the CubeStar is twofold: firstly the attitude estimation of the CubeStar in the emulation environment can be investigated with captured images that are undistorted, and secondly the distortion coefficients that are determined by the calibration procedure can be used for a predistortion method that is also discussed in this section. For the CubeStar to autonomously estimate its attitude, projected stars can be predistorted to allow the CubeStar to observe the stars at undistorted locations and successfully identify them.

5.1 Design

In this section an overview of the physical aspects of the emulation environment that was built for HIL experiments are discussed. The emulation environment consists of 6 parts:

- The CubeStar star tracker with its required electronic developer board.
- The cardboard box used as the frame of the physical environment.

- A black cloth wrapped inside of the frame.
- A computer monitor for the projected stars.
- A custom stand designed in this project for the star tracker.
- A secondary lens with a custom designed holder.

To keep the environment low cost, the frame consist of a cardboard box that blocks out most of the external light. The box also serves as a frame to keep a black fabric cloth fixed inside the environment. The cloth serves two purposes: firstly to block out any additional light that may penetrate the cardboard frame, and secondly the cloth minimizes any reflections that may occur inside the environment from the only light source: the computer monitor.

The computer monitor used inside the environment is a 23-inch (58,42 cm) Dell P2314H LCD monitor that has a Light-Emitting Diode (LED) backlight, the backlight is edge-lit at the bottom of the monitor. The monitor has a pixel resolution of 1920 pixels (horizontally) by 1080 pixels (vertically). The main reason for using a LCD monitor is that there was already one available in the Electronic Systems Laboratory (ESL). LCD monitors are much less expensive than the alternative, Organic Light-Emitting Diode (OLED) monitors.

The star tracker must be placed with the centre of its image plane aligned with the centre of the monitor. To achieve this, a stand for the star tracker was designed in Autodesk Inventor. The stand has a platform on which the star tracker and its developer electronics can be placed, the platform can move vertically and horizontally to make finer adjustments to align the CubeStar correctly. The design of the platform can be seen in figure 5.1, it was built out of plywood since it is inexpensive and rigid enough for the application.

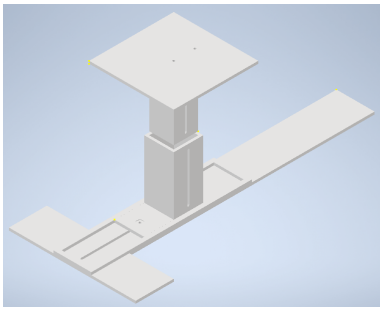


Figure 5.1: Star Tracker Stand Design

5.1.1 Minimum Distance From Monitor

To determine the minimum distance that the star tracker should be placed from the monitor for the entire projected image to be present on the image plane, the properties of the pinhole camera model discussed in section 3.1.3 are used. The calculations are assisted by figure 5.2, which is an adapted version of figure 3.2:

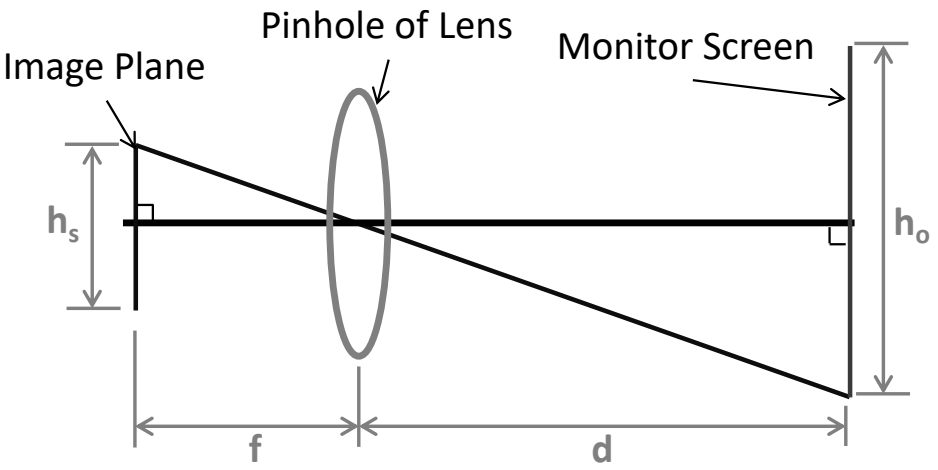


Figure 5.2: Similar Triangles Pinhole Camera Model

From figure 5.2 and equation 3.13, it can be shown that

$$\begin{aligned} d &= \frac{f\frac{h_o}{2}}{\frac{h_s}{2}} \\ &= \frac{fh_o}{h_s} \end{aligned} \tag{5.1}$$

where
d: distance of object to lens in mm
f: focal length of lens in mm
h_o: height of physical object that is captured in mm
h_s: height of image sensor in mm

The sensor size of the CubeStar engineering model used for this project is approximately 6.9mm in width and 5.5mm in height, the value h_s can therefore be set to 5.5mm. To obtain the values for d and h_o, the pixel pitch values of the monitor is first required. The height of the monitor is measured as approximately 287mm and it contains 1080 pixels vertically, the vertical pixel pitch $p_{ymonitor}$ of the monitor's pixels can therefore be calculated as

$$\begin{aligned} p_{ymonitor} &= \frac{287\text{mm}}{1080pixels} \\ &= 0.2657\text{mm}/pixel \end{aligned} \tag{5.2}$$

Since the projected image will have the same vertical amount of pixels as the image sensor (1024 pixels), the height h_o can be calculated as

$$\begin{aligned} h_o &= p_{ymonitor} \cdot 1024 \\ &= 0.2657\text{mm} \cdot 1024 \\ &= 272.08\text{mm} \end{aligned} \tag{5.3}$$

If the star tracker is positioned to align its image sensor's centre with the centre of the monitor, equation 5.1 can be used with a lens focal length of approximately 6mm to calculate the distance d as

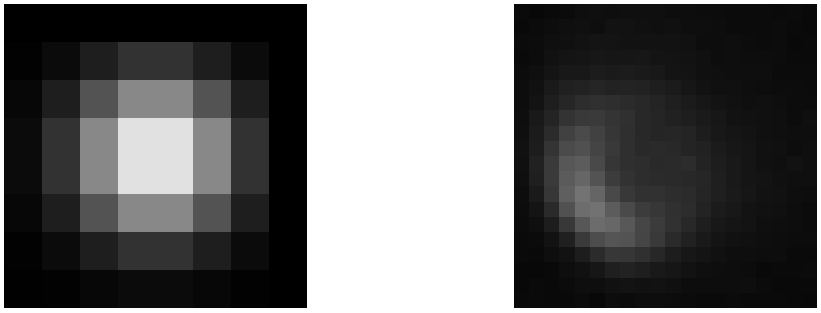
$$\begin{aligned} d &= \frac{fh_o}{h_s} \\ &= \frac{6 * 272.08}{5.5} \\ &= 296.81\text{mm} \end{aligned} \tag{5.4}$$

The star tracker must therefore be placed on the platform at a minimum distance of 296.81mm from the monitor to ensure that the entire projected image will be present on the image plane of the star tracker.

5.1.2 Star Tracker Focus Adjustment

For project objective 5, the environment must be suitable for star trackers that have already been calibrated as well as star trackers that have not yet been calibrated. Star trackers that have not been calibrated can be focused on the monitor in the emulation environment, however calibrated star trackers are focussed at infinity. The star tracker can then be considered to have a "fixed-focus", this implies that since the star tracker's lens does not have autofocus any object that is closer to the star tracker than the infinity focus distance will not be in focus. If the lens of a calibrated star tracker is refocussed, all distortion coefficients are invalid and the star tracker will need to be recalibrated. Therefore, a calibrated star tracker placed in the emulation environment will not be able to focus on the monitor, resulting in blurred images. The approach of this project to attempt to solve this problem is to use a secondary lens to bring objects on the monitor back into focus for a star tracker that is focussed at infinity to allow a calibrated star tracker to be used in the environment.

Figure 5.3a below shows the star "Gamma Crux" that is generated, figure 5.3b shows the same star after it was captured by the CubeStar in the emulation environment without a secondary lens. It is visibly blurry with a size of approximately 20x20 pixels, this is much larger than the generated star kernel size of 7x7 pixels. Before a suitable lens was chosen, a brief investigation on the optics of camera lenses was performed which is discussed in the following section.



(a) Generated Star (b) Captured Star

Figure 5.3: Star 60718 ("Gamma Crux") Generated And Captured With No Secondary Lens

5.1.2.1 Optics Of Lenses

Figure 5.4 below shows a basic illustration of the impact that the primary lens of a camera has on light rays when the camera observes objects at infinity. It can be assumed that light rays from objects at infinity will enter the camera lens parallel to each other, if the lens of the camera is focussed at infinity it will then converge the parallel light rays to a single point onto the sensor plane to produce clear objects on the image. The lens of the CubeStar contains a convex lens, implying that it is thicker at its centre than at its edges to converge light rays.

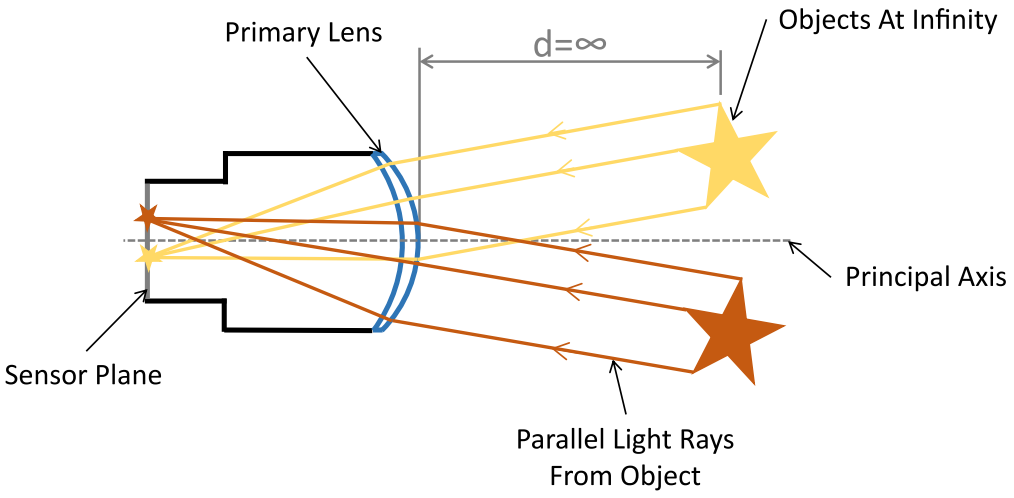


Figure 5.4: Camera Optics: Objects At Infinity

When an object is closer to the lens than infinity, the light rays from the object will not enter the camera lens parallel to each other and therefore the convex primary lens will not converge the light rays enough to converge to a single point on the sensor plane, instead the point of convergence (also known as the focal point) will be behind the sensor plane and therefore the object on the image will not be in focus. This effect is illustrated in figure 5.5 below with two objects: one object is at an infinite distance and therefore its light rays enter the lens parallel

and converges to a single point to produce a clear object on the image, another object is located closer to the lens than infinity, the light rays from this object does not enter the lens parallel to each other and therefore they converge to a point behind the sensor plane resulting in a blurred object on the image.

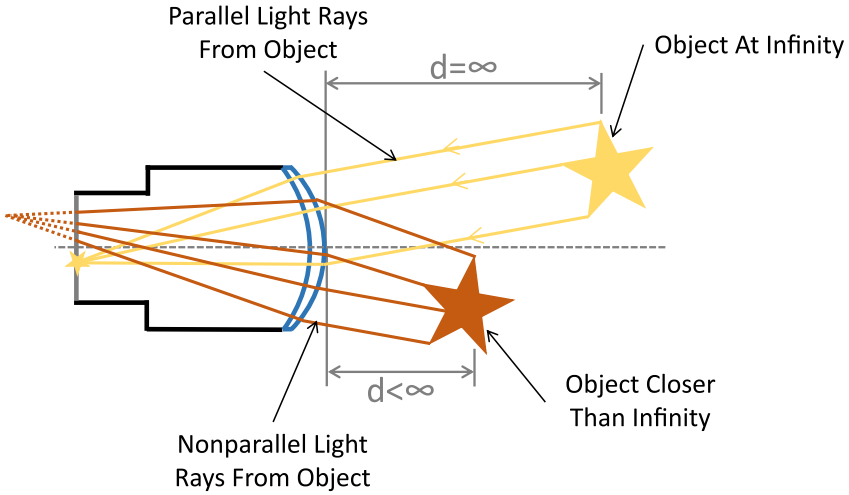


Figure 5.5: Camera Optics: Objects Closer Than Infinity Without Secondary Lens

To move the point of convergence back onto the sensor plane for objects that are placed closer to the camera lens than infinity, a secondary convex lens is used to compensate for the primary lens that is not converging the light rays enough. A suitable secondary lens will converge the light rays from objects at a certain distance from it so that they exit the secondary lens parallel to each other, these parallel light rays then enter the camera's primary lens and is converged to a single point on the sensor plane to produce clear objects on the image. The secondary lens therefore acts as a collimator lens. This effect is illustrated in figure 5.6 below with two objects placed closer to the lens than infinity.

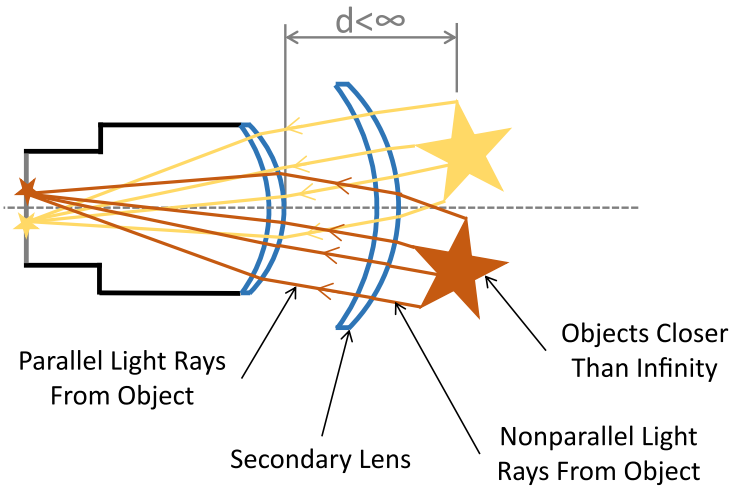


Figure 5.6: Camera Optics: Objects Closer Than Infinity With Secondary Lens

The problem at hand is very similar to that of a person with far-sightedness, the muscles of the eye does not form the lens of the eye to be convex enough and therefore the focal point of objects are behind the retina. Optometrist then use secondary convex lenses in glasses to compensate for the lack of convergence. Since this project faced the same problem, an optometry lens kit was acquired from a local optometrist. The kit (as shown below in figure 5.7) consist of convex and concave lenses in increments of $0.25 dioptres$ and is used to iteratively determine what lenses should be used in a pair of glasses for a person that does not have optimal eyesight.

Figure 5.7: Optometry Lens Kit

Convex lenses cause light rays to converge and are measured in positive dioptres, concave lenses cause light rays to diverge and are measured in negative dioptres. A dioptre (D) is the unit of measure for the optical power of a lens and is defined as the reciprocal value of the lens' focal length in metres, as shown in equation 5.5 below.

$$D = \frac{1}{f} \tag{5.5}$$

where

f: focal length of lens in metres
D: optical power of lens in dioptres

5.1.2.2 Determining A Suitable Lens For CubeStar

With the basic knowledge from the previous section, a secondary lens for the emulation environment could be determined. Since the only available star tracker for this project was an engineering model of the CubeStar, the focus adjustment was only performed on the CubeStar used in this project.

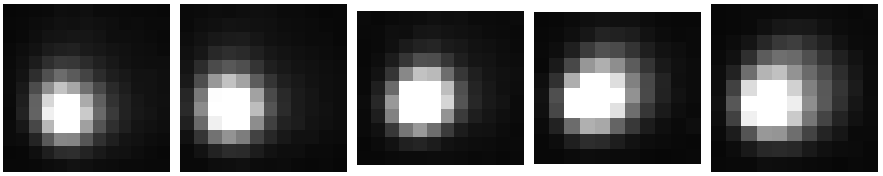
Various convex lenses were tested from the kit by placing it in front of the CubeStar in a 3D-printed holder that was designed in Autodesk Inventor. Since each lens has its own focal point at a certain distance based on its optical power, the CubeStar also had to be moved to the appropriate distance from the monitor. In section 5.1.1 it was determined that the CubeStar must be placed at a minimum distance of 296.81mm from the monitor (from equation 5.4) for the entire projected image to be present on the image plane. Moving the CubeStar closer to the monitor than the minimum distance will result in some of the projected image being outside of the CubeStar's FoV. To prevent this, firstly the maximum dioptre lens was determined.

A 3.5*dioptre* lens has a focal distance of 285.71mm (as calculated in equation 5.6), which is smaller than the minimum distance from the monitor, the next available lens is a 3.25*dioptre* lens with a focal distance of 307.69mm (as calculated in equation 5.7). Therefore, a lens with an optical power greater than 3.25*dioptre* can not be used as the CubeStar will then need to be closer to the monitor than the minimum distance (as per equation 5.4).

$$\begin{aligned} f_{3.5D} &= \frac{1}{3.5D} \\ &= 285.71\text{mm} \end{aligned} \tag{5.6}$$

$$\begin{aligned} f_{3.25D} &= \frac{1}{3.25D} \\ &= 307.69\text{mm} \end{aligned} \tag{5.7}$$

Each lens from 1.75*dioptre* to 3.25*dioptre* was tested iteratively by moving the CubeStar to the appropriate distance from the monitor and taking an image of the Southern Cross constellation projected on the monitor. The images were analysed to find the lens that delivered the most clear stars. Figure 5.8 shows star "60718"/"Gamma Crux" with different lenses, it was observed that all stars had a slight trail towards the top right caused by the bottom mounted edge-lit backlight of the LCD monitor, however some lenses delivered bigger trails than others.



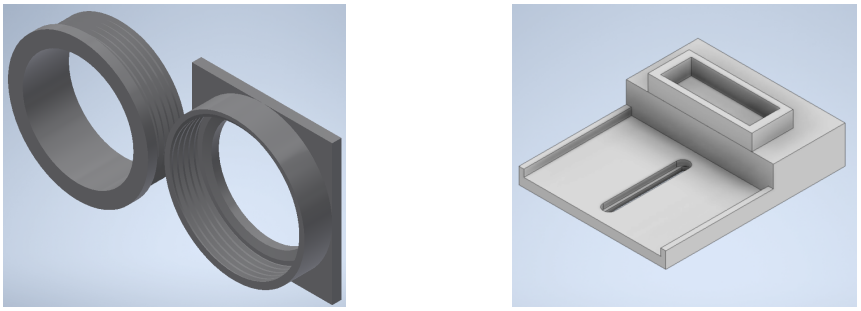
(a) +1.75D (b) +2D (c) +2.5D (d) +3D (e) +3.5D

Figure 5.8: Star 60718 Captured With Secondary Lens

Figure 5.8 illustrates that the star sizes decreases from the 1.75*dioptre* lens to the 3*dioptre* lens, and then starts increasing again. Upon inspection, the 3*dioptre* lens was chosen as it consistently delivered stars with the best focus and shortest trails compared to other lenses. For the 3*dioptre* lens, the distance that the CubeStar must be placed from the monitor is calculated as

$$\begin{aligned} d_{3D} &= \frac{1}{3} \\ &= 333.33\text{mm} \end{aligned} \tag{5.8}$$

Since the optometry kit had to be returned, a 3*dioptre* lens was manufactured for the project and sponsored by "Lambert Fick & Maree Optometrists". The lens is coated with an anti-reflective layer and is 65mm in diameter. Two final parts were designed in Autodesk Inventor for the emulation environment, a holder for the secondary lens and a holder for the CubeStar to allow the secondary lens to be positioned in front of the CubeStar's primary lens. Both designs can be seen in figure 5.9, the parts were 3D printed and placed on the star tracker stand inside the emulation environment.



(a) Secondary Lens Holder Design (b) CubeStar and Lens Holder

Figure 5.9: CubeStar and Lens Holders 3D Print Designs

The final version of the emulation environment is shown in figures 5.10a to 5.10d below. The total size of the emulation environment is 0.8m × 0.6m × 0.5m, which satisfies project outcomes 3. The figures show the cardboard box that is used as a frame, with the black cloth wrapped on the inside of the frame. The stand that was built can be seen on the inside of the frame with the CubeStar placed on it along with the secondary lens. The monitor can also be seen with a white background, the white background was required to produce enough light to

show the inside of the emulation environment as the black cloth makes the inside of the emulation environment pitch black.

(a) Top View

(b) Front View

(c) Inside View

(d) CubeStar with secondary lens placed on platform

Figure 5.10: Emulation Environment Setup

5.2 Analysis

As expected, images captured by the CubeStar of projected stars on the monitor had a large amount of errors, these errors were caused by the lens distortion as well as a misalignment between the CubeStar and the monitor caused by the placement of the CubeStar. The methods to minimize these errors and the results are presented in this section.

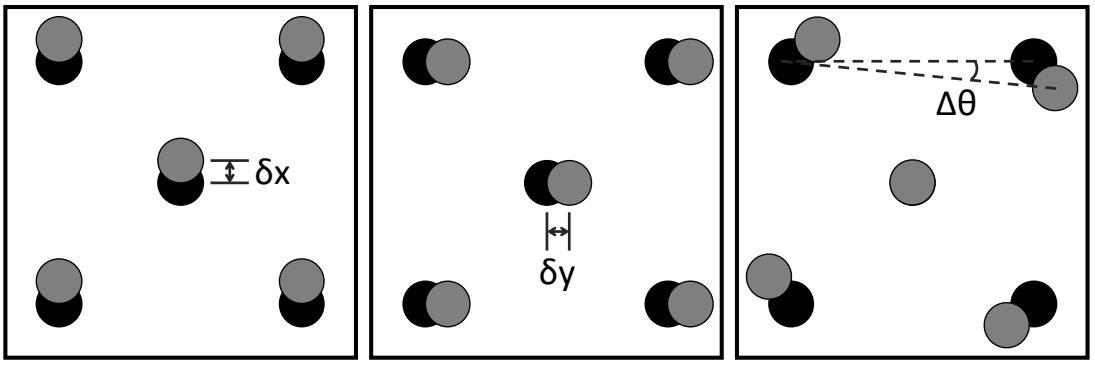
5.2.1 Alignment Correction

Before the star tracker can capture accurate star images, it must first be aligned properly with the monitor with two constraints. Firstly, the monitor must be

parallel to the CubeStar's image plane and secondly, the CubeStar's boresight must point to the centre of the monitor. Any alignment offset will cause large errors that are superimposed on the errors caused by the lens distortion. The five alignment offsets to be compensated for are:

- Vertical offset
- Horizontal offset
- Rotational offset
- Tilt offset
- Swivel offset

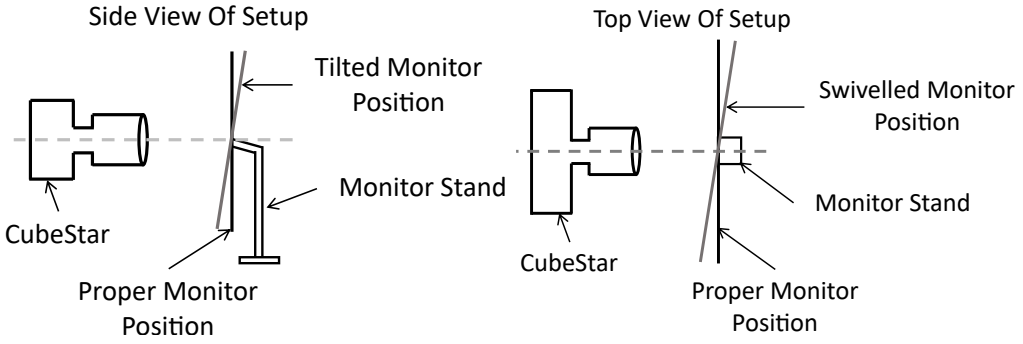
The effect of vertical-, horizontal- and rotational offsets are illustrated in figure 5.11 below. These offsets are caused by the boresight of the CubeStar that does not point to the centre of the monitor.



(a) Vertical Offset (b) Horizontal Offset (c) Rotational Offset

Figure 5.11: Offsets Caused By CubeStar Boresight Misalignment. The black dots represent generated stars and the grey dots represent captured stars.

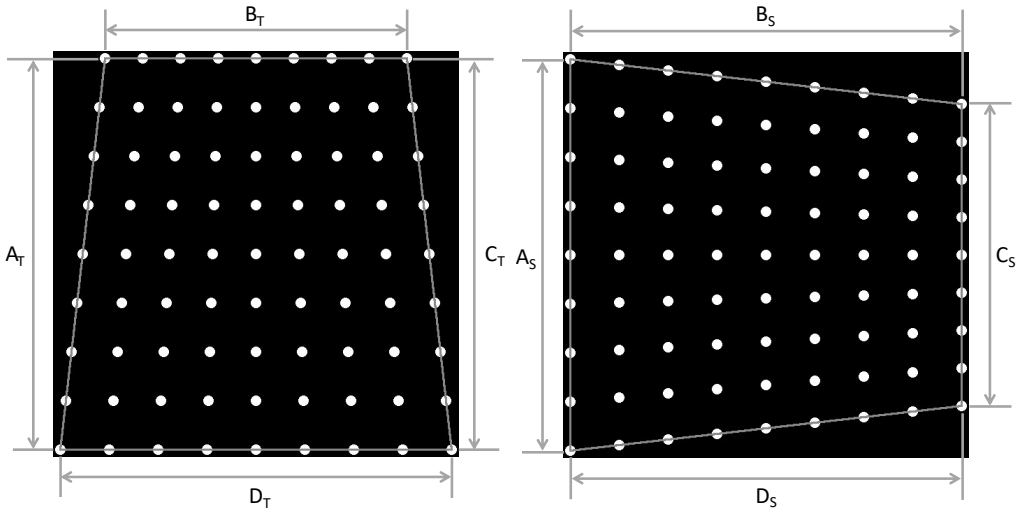
A misalignment between the CubeStar's image plane and the monitor that causes tilt and swivel offsets are illustrated in figure 5.12 below. A tilt offset is present when the monitor is rotated about its horizontal axis and a swivel offset is present when the monitor is rotated about its vertical axis. These offsets result in the captured image not being symmetrical about the centre of the image.



(a) Tilted Monitor (b) Swivelled Monitor

Figure 5.12: Misalignment Between Monitor And CubeStar

To determine whether there are tilt or swivel offsets present, a symmetrical alignment pattern can be projected onto the centre of the monitor. For the emulation environment in this project, a symmetrical square grid of stars is used as the alignment pattern. With no tilt or swivel offset present, the alignment pattern will have four sides that must be the same length irrespective of the radial distortion that is present. The effect of tilt- and swivel offsets are illustrated in figure 5.13 with the simulated result of a captured alignment pattern with a tilt or swivel offset present. From figure 5.13a, a tilt offset will cause the length of the top and bottom sides to differ such that $B_T \neq D_T$. From figure 5.13b, a swivel offset will cause the length of the left and right sides to differ such that $A_S \neq C_S$.



(a) Effect of Tilt Offset (b) Effect of Swivel Offset

Figure 5.13: The Simulated Effect Of Tilt- And Swivel Offsets On A Captured Alignment Pattern

It was observed that at the distance the CubeStar is placed from the monitor (333.33mm as per equation 5.8), the star tracker is very sensitive to adjustments when trying to compensate for vertical-, horizontal- and rotational offsets by physically adjusting the position of the star tracker. Therefore, once the CubeStar was

placed with the vertical- and horizontal offsets minimized to below 10 pixels, the rest of the vertical- and horizontal offsets, as well as the rotational offset, could be compensated for in the software that projects the image. This compensation is implemented by measuring the distance that the measured centroid of the bore-sight pointing star lies from the centre of the image in the x- and y-axis (δx and δy from figure 5.11). δx and δy is then added to the centroid of each star that is projected. For the rotational offset the angle ($\Delta\theta$ from figure 5.11c) between the centroids of the top left star and the top right star is calculated in radians. Each star's image plane coordinate location is then converted to polar coordinates such that

$$[X_C, Y_C] = |Z| \angle \theta \tag{5.9}$$

with

$$|Z| = |\sqrt{(X_C - X_O)^2 + (Y_C - Y_O)^2}| \tag{5.10}$$

$$\theta = \arctan2(\frac{-(Y_C - Y_O)}{X_C - X_O}) \tag{5.11}$$

where
X_C: centroid of star on the x-axis in pixels
Y_C: centroid of star on the y-axis in pixels
X_O: centre of image on the x-axis in pixels
Y_O: centre of image on the y-axis in pixels
$|Z|$: distance from star centroid to centre of the image in pixels
θ: polar angle of star in radians

Note that in equation 5.11 the function "arctan2" calculates the four-quadrant inverse tangent of the two input arguments. As the function assumes a standard Cartesian plane that has a vertical y-axis that increases upward, the negative sign in the numerator of equation 5.11 reverses the sign of the value to compensate for the monitor image plane that has a vertical y-axis that increases downwards.

The rotational offset $\Delta\theta$ can then be added to each star's polar angle θ and the new centroid of each star is calculated as

$$\begin{aligned} X_{rot} &= |Z| \cos(\theta + \Delta\theta) \\ Y_{rot} &= |Z| \sin(\theta + \Delta\theta) \end{aligned} \tag{5.12}$$

where
X_{rot}: rotated star centroid on the x-axis in pixels
Y_{rot}: rotated star centroid on the y-axis in pixels

With the vertical-, horizontal- and rotational offsets removed, the tilt- and swivel offsets could be determined by evaluating the length of the sides illustrated in figure 5.13. To minimize the tilt offset, sides B_T and D_T must be the same length and to minimize the swivel offset, sides A_S and C_S must be the same length. The

two offsets are minimized by iteratively tilting and swivelling the monitor and evaluating the side lengths until they are as close to one another as possible. Once the monitor is moved, the vertical-, horizontal- and rotational offsets will change and needs to be determined again. This is a very time consuming process, but only have to be repeated once before a set of images can be captured. The alignment correction procedure that is followed to minimize all five offsets is summarized below:

1. Project and capture the alignment pattern in the centre of the monitor.
2. Determine the horizontal- and vertical offset between the centre star's centroid and the centre of the image.
3. Adjust CubeStar's position to compensate for the horizontal- and vertical offset.
4. Repeat steps 1 to 3 until the offsets are below 10 pixels.
5. Compensate for the remaining horizontal and vertical offset in the software as described above.
6. Determine the tilt offset and adjust the monitor to compensate for it.
7. Repeat steps 1 to 6.
8. Determine the swivel offset and adjust the monitor to compensate for it.
9. Repeat steps 1 to 8.

5.2.2 Alignment Correction Results

Even with lens distortion present, the effectiveness of the alignment correction procedure could be evaluated. Since the distortion is mainly radially symmetrical about the centre of the image, the four sides of the symmetrical alignment pattern that is displayed should still be equal to one another. The alignment pattern that is projected onto the monitor for the CubeStar's alignment correction is a square grid of 15 by 15 stars with a spacing of 50 pixels between the star centroids of each row and column, figure 5.14 shows a scaled down and cropped version of this alignment pattern. The generated distance of each side, d_{sides}, measured between the corner star centroids, is calculated as

$$\begin{aligned} d_{sides} &= 50pixels * (15-1)rows \\ &= 700pixels \end{aligned} \tag{5.13}$$

The two diagonal distances between the four corner stars are calculated with the Pythagorean Theorem as

$$\begin{aligned} d_{diagonal} &= \sqrt{(50pixels * (15-1)rows)^2 + (50pixels * (15-1)columns)^2} \\ &= 989.95pixels \end{aligned} \tag{5.14}$$

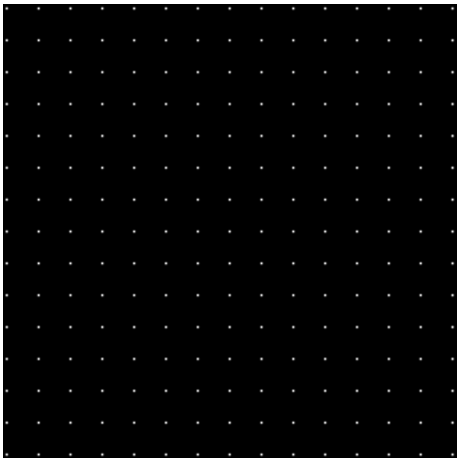


Figure 5.14: Scaled Down And Cropped Alignment Pattern

With radial distortion present and no alignment offsets the centre star must lie in the centre of the image and the length of the 4 sides should be equal to one another. It will not, however, be equal to the distance calculated in equation 5.13 as the distance will be shorter and depend on the intensity of the radial distortion present. Similarly, the two diagonal lengths should also be equal to one another but will not be equal to the distance calculated in equation 5.14.

Figure 5.15 shows a captured grid after the alignment procedure was performed and the remaining offsets were compensated for in the software. For this image, the star tracker could be positioned with a centre star centroid of $(s_{113X}, s_{113Y}) = (639.98, 519.03)$, this results in a x-axis horizontal offset of 0.02 pixels and an y-axis vertical offset of 0.03 pixels. The rotational offset was minimized to 9.1 arcseconds. To show the remaining tilt- and swivel offsets that are present, the results of the corner star distances are summarized in table 5.1, with the notations indicated in figure 5.15.

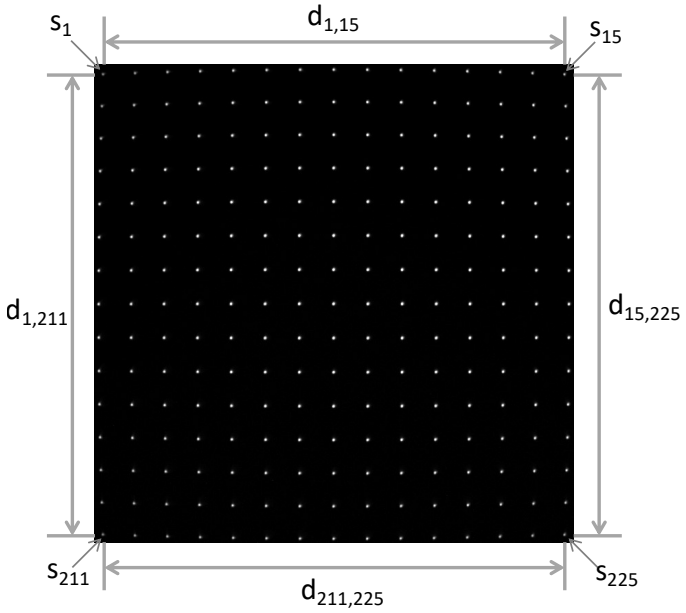


Figure 5.15: Captured Alignment Pattern After Alignment Procedure

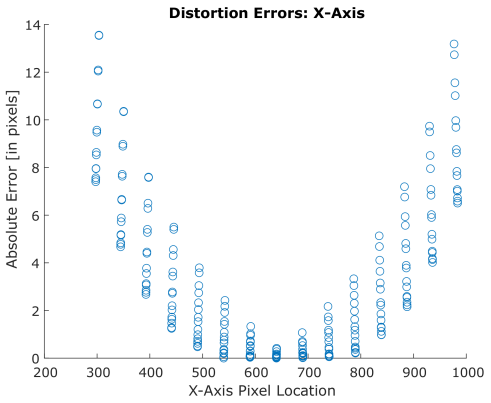
Table 5.1: Captured Alignment Pattern Side Lengths Results

Distance Label	Distance in pixels
$d_{1,211}$	672.74
$d_{1,15}$	673.71
$d_{15,225}$	673.35
$d_{211,225}$	673.28
$d_{1,225}$	952.22
$d_{15,211}$	952.08
$\lvert d_{1,15} - d_{211,225} \rvert$	0.43
$\lvert d_{1,211} - d_{15,225} \rvert$	0.61
$\lvert d_{1,225} - d_{15,211} \rvert$	0.14

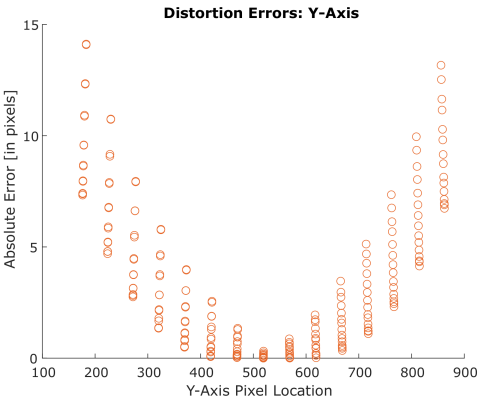
The last three entries from table 5.1 show the difference between the length of the top and bottom sides, the left and right sides, and the two diagonal sides. Although each alignment correction process will yield different results, it was decided that once these values could be minimized to below $1pixel$ it shows that the tilt- and swivel offset is minimized to an acceptable amount and the calibration of the star tracker could be performed.

For the captured alignment pattern in figure 5.15, the errors between the generated star centroids and the measured star centroids are shown in figure 5.16. Since the alignment errors are removed to an acceptable amount, the remaining errors show the effect of the lens distortion on the star centroids. Radial distortion is quadratic which implies that its effect increases as the squared distance to the centre of distortion increases. This is clearly observed in the results shown in figure

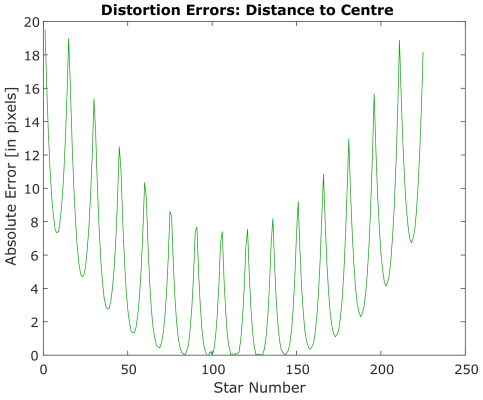
5.16 and therefore a calibration of the star tracker is required to find the distortion coefficients that will undistort the star centroids to reduce these distortion errors.



(a) X-Axis



(b) Y-Axis



(c) Distance To Centre

Figure 5.16: Square Grid Distortion Errors

The average centroid errors and the standard deviation from the average that was observed from figure 5.16 for each axis and the distance to the centre is shown in the table below, along with the maximum centroid error of each axis and the maximum observed distance from the centre error. These values are extremely large and will be used to compare to the accuracy produced by the distortion correction once the calibration is completed.

Table 5.2: Distorted Centroid Error Results

	X-Axis	Y-Axis	Distance to Centre
Average Error [in pixels]	3.4427	3.5448	5.2526
Standard Deviation [in pixels]	3.4107	3.4675	4.5155
Maximum Error [in pixels]	13.5547	14.1205	19.5484

5.2.3 CubeStar Calibration

To remove the radial distortion errors from the measured star centroids, the radial distortion model discussed in section 3.2.4 is used. To apply the distortion model to image points, equations 3.19 and 3.20 are used to remove the distortion from the image points. However, to use this radial distortion model a calibration method is required to estimate the radial distortion coefficients K_1 to K_6. As mentioned in the chapter introduction, the calibration of the CubeStar is twofold: the distortion coefficients are used in the radial distortion model to undistort image points, and the distortion coefficients are used in a predistortion model that is discussed in section 5.2.5. From here the text will refer to the projected star centroids as the "generated centroids" and the centroids of the stars that were observed as the "measured centroids".

Various techniques exist to calibrate star trackers. Those that were investigated for this project include: 1) the plumb line approach [21],[32] where lines that are known to be straight are distorted to circular arcs and used to solve for the distortion coefficients, 2) capturing a chequered pattern from different angles and using Matlab's Camera Calibration Toolbox to solve for the coefficients [6],[33], 3) capturing star constellations and using software by CubeSpace to solve for the coefficients [23], 4) using a collimated light source placed in front of the star tracker [34] and 5) using a projector setup to display calibration patterns for the star tracker to capture [35]. Most of these calibration techniques have drawbacks that make them not ideal for this project, as the star tracker is focussed at infinity the chequered pattern approach would be unsuitable since a very large chequered-board would be required, projector setups requires a large open area which was not available in the ESL and projectors also come with their own projection distortion, and the ESL has no collimated light source available for use in this project. While the software written by CubeSpace presented no apparent disadvantage except that it could become time consuming to manually assign the stars that were observed to their catalogue star, it was decided to use a calibration technique more specific to the CubeStar in the emulation environment.

In the emulation environment the coordinates whereto a star is projected is also where a star is expected to be observed on the sensor image plane with no distortion present. Therefore, the generated centroid of each star can be used as the undistorted centroid location of its distorted measured counterpart. With a set of undistorted centroids and their matching distorted centroids, a Least-Squares (LS) minimization technique is used to estimate the distortion coefficients of the radial distortion model. To get a set of generated and measured centroids, the alignment pattern that is used to align the CubeStar (as discussed in the previous section) is now used as a calibration pattern, the pattern that was captured by the CubeStar can be seen in figure 5.15.

Once each star in the calibration pattern is detected and its centroid calculated, a bubble sorting algorithm is performed on the generated- and measured centroids for each row of stars to assign a number to each star, this is the matching method

used to ensure each generated star centroid is matched with its correct measured centroid counterpart. Figure 5.17 shows a scaled down and cropped version of the measured centroids after each star is assigned a number.

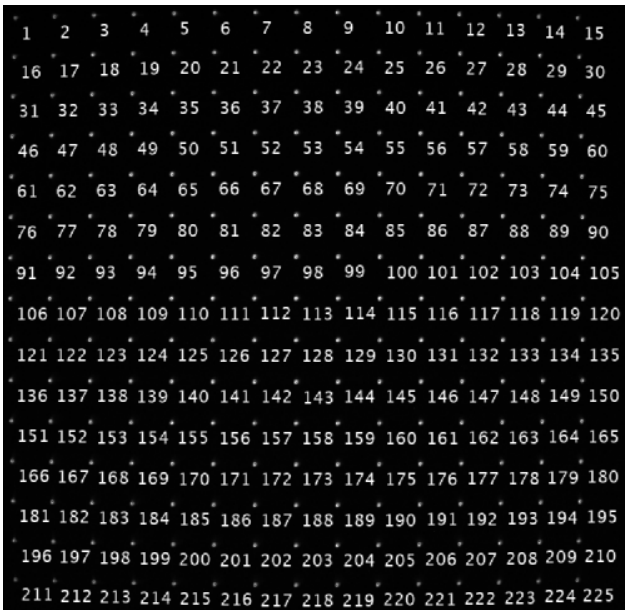


Figure 5.17: Captured Square Grid With Numbered Stars

5.2.3.1 LS Method To Estimate Distortion Coefficients

The LS method to estimate the distortion coefficients is presented in this section by first defining the variables that are used and then setting up the equations for the LS minimization.

Variable Definitions:
$x_u(i)$, $y_u(i)$: generated (undistorted) coordinates of star i
$x_d(i)$, $y_d(i)$: measured (distorted) coordinates of star i
x_o, y_o: principal point of optics (centre of distortion)
$r_d(i)$: distance from distorted point to principal point
$e_x(i)$: distortion measurement error in x-direction
$e_y(i)$: distortion measurement error in y-direction

Radial Distortion Model: (From 3.19 and 3.20)

$$x_u(i) - x_o = (x_d(i) - x_o) * (1 + K_1 * r_d(i)^2 + K_2 * r_d(i)^4 + K_3 * r_d(i)^6) \quad (5.15)$$
$$y_u(i) - y_o = (y_d(i) - y_o) * (1 + K_4 * r_d(i)^2 + K_5 * r_d(i)^4 + K_6 * r_d(i)^6) \quad (5.16)$$
$$with\ r_d(i)^2 = (x_d(i) - x_o)^2 + (y_d(i) - y_o)^2$$

LS measurement vector derivation for e_x: (From 5.15)

$$Declare\ x_t(i) = (x_d(i) - x_o) \tag{5.17}$$

$$x_u(i) - x_o = (x_d(i) - x_o) + x_t(i) * K_1 * r_d(i)^2 + x_t(i) * K_2 * r_d(i)^4 + x_t(i) * K_3 * r_d(i)^6$$

$$x_u(i) - x_o - (x_d(i) - x_o) = x_t(i) * K_1 * r_d(i)^2 + x_t(i) * K_2 * r_d(i)^4 + x_t(i) * K_3 * r_d(i)^6$$

$$x_u(i) - x_d(i) = x_t(i) * K_1 * r_d(i)^2 + x_t(i) * K_2 * r_d(i)^4 + x_t(i) * K_3 * r_d(i)^6 \tag{5.18}$$

$$Then\ e_x(i) = x_u(i) - x_d(i) \tag{5.19}$$

$$and\ e_x(i) = [x_t(i) * r_d(i)^2 \quad x_t(i) * r_d(i)^4 \quad x_t(i) * r_d(i)^6] * \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix} = \boldsymbol{\varphi_x^T(i).\theta_x} \tag{5.20}$$

where

$\boldsymbol{\varphi_x^T(i)}$: x-axis regression vector of LS problem

$\boldsymbol{\theta}_x$: unknown parameter vector for x-axis

The same procedure is performed for e_y with equation 5.16 to find the regression vector $\boldsymbol{\varphi_y^T(i)}$ and parameter vector $\boldsymbol{\theta_y}$ for the y-axis.

LS measurement n-element vector (from 5.19):

$$\boldsymbol{X} = \begin{bmatrix} e_x(1) \\ e_x(2) \\ \dots \\ e_x(n) \end{bmatrix} \tag{5.21}$$

$$\boldsymbol{Y} = \begin{bmatrix} e_y(1) \\ e_y(2) \\ \dots \\ e_y(n) \end{bmatrix} \tag{5.22}$$

Regression nx3 matrix from (5.20):

$$\boldsymbol{C_x} = \begin{bmatrix} \boldsymbol{\varphi_x^T}(1) \\ \boldsymbol{\varphi_x^T}(2) \\ \dots \\ \boldsymbol{\varphi_x^T}(n) \end{bmatrix} \tag{5.23}$$

$$\boldsymbol{C_y} = \begin{bmatrix} \boldsymbol{\varphi_y^T}(1) \\ \boldsymbol{\varphi_y^T}(2) \\ \dots \\ \boldsymbol{\varphi_y^T}(n) \end{bmatrix} \tag{5.24}$$

Best LS estimated parameter vector from Batch Method:

$$\begin{aligned} \hat{\boldsymbol{\theta}}_x &= [\boldsymbol{C_x^T C_x}]^{-1} \boldsymbol{C_x^T X} \\ &= [K_1 \quad K_2 \quad K_3] \\ &= \boldsymbol{\theta}_x^T \end{aligned} \tag{5.25}$$

$$\begin{aligned} \hat{\boldsymbol{\theta}}_y &= [\boldsymbol{C_y^T C_y}]^{-1} \boldsymbol{C_y^T Y} \\ &= [K_4 \quad K_5 \quad K_6] \\ &= \boldsymbol{\theta}_y^T \end{aligned} \tag{5.26}$$

Equations 5.27 and 5.28 below can then be used to undistort an image point i.

$$x_u(i) = (x_d(i) - x_o) * (1 + K_1 * r_d(i)^2 + K_2 * r_d(i)^4 + K_3 * r_d(i)^6) + x_o \tag{5.27}$$

$$y_u(i) = (y_d(i) - y_o) * (1 + K_4 * r_d(i)^2 + K_5 * r_d(i)^4 + K_6 * r_d(i)^6) + y_o \tag{5.28}$$

To find the optimal centre of distortion, also referred to as the optimal principal point, an iterative loop is performed to find the centre of distortion that yields the

smallest undistortion errors. The starting centre of distortion is set to the centre of the image, the value is then iteratively changed in increments of 0.1 pixels, with a limit of 10 pixels to the left and right in both the x- and y-axes.

5.2.4 CubeStar Calibration Results

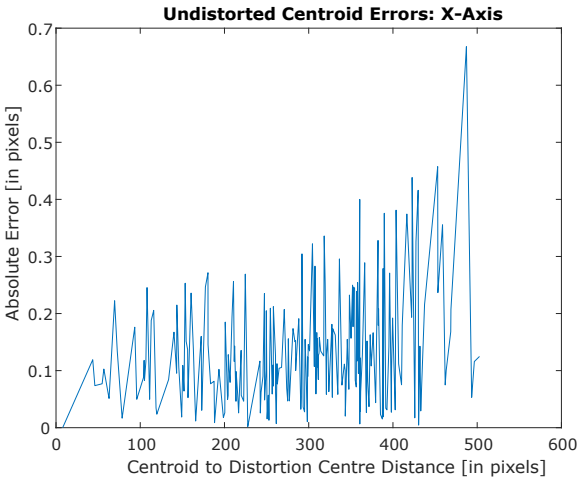
With each generated centroid in the calibration pattern matched with its measured centroid counterpart, the x- and y-axis generated centroid values are stored in x_u and y_u respectively as the undistorted centroid locations and the x- and y-axis measured centroid values are stored in x_d and y_d respectively as the distorted centroid locations. The LS method described in the previous section is then implemented with a Matlab script to estimate the distortion coefficients K_1 to K_6 for the radial distortion model in equations 3.19 and 3.20. For the captured calibration pattern in figure 5.15 the estimated distortion coefficients are shown in table 5.3 below.

Table 5.3: Estimated Distortion Coefficients

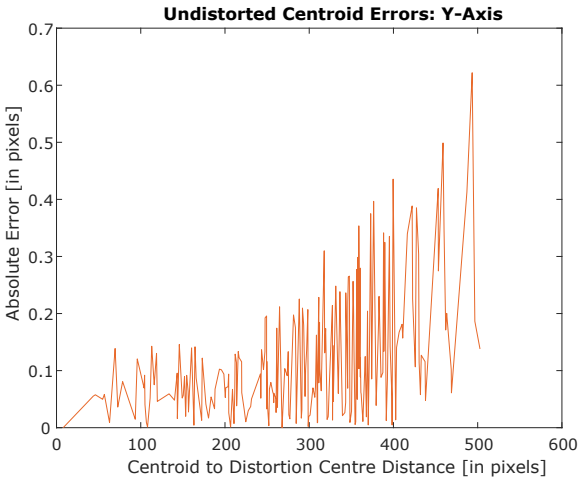
X-Axis		Y-Axis	
K_1	$1.3804 * 10^{-7}$	K_4	$1.5395 * 10^{-7}$
K_2	$4.0888 * 10^{-13}$	K_5	$2.6587 * 10^{-13}$
K_3	$-1.1505 * 10^{-18}$	K_6	$-7.4120 * 10^{-19}$

To determine the accuracy that the radial distortion model with this calibration method can provide, three values are evaluated: the absolute x- and y-axis centroid errors and the absolute distance errors. The x- and y-axis centroid error for each star is defined as the error between the undistorted centroid and the generated centroid for each respective axis. The distance error is defined as the error between the distance of the undistorted centroid to the distortion centre and the distance of the generated centroid to the image centre. Since the accuracy of the GVA that identifies the stars is greatly dependant on the distance between the star centroids, evaluating the undistorted distance errors provides valuable insight to the accuracy that the distortion model will in fact provide.

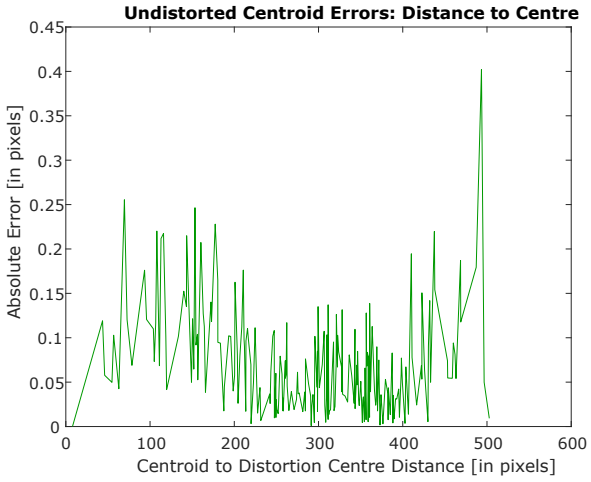
Before these errors can be determined, each distorted star centroid in the captured calibration pattern is undistorted with equations 5.27 and 5.28. The errors of each centroid are illustrated in figure 5.18, which show the errors of each centroid for the x- and y-axis, and the distance errors of each centroid. For illustration purposes, the errors are ordered by increasing distance from each centroid to the centre of distortion. From the undistorted centroid errors in figures 5.18a and 5.18b it is observed that for each axis the error of the undistorted centroid increases as the distance to the centre of the image increases, which is as expected due to the quadratic nature of radial distortion.



(a) X-Axis



(b) Y-Axis



(c) Distance to Centre

Figure 5.18: Undistortion Errors Results

For the results in figure 5.18, the average absolute error for each axis and the average absolute distance error are summarized in table 5.4 below, along with the standard deviation of these averages, the maximum value for each error and the optimal principal point.

Table 5.4: Undistortion Errors Results

	X-Axis	Y-Axis	Distance to Centre
Average Error [in pixels]	0.1332	0.1154	0.0711
Standard Deviation [in pixels]	0.1003	0.1064	0.0579
Maximum Error [in pixels]	0.6676	0.6219	0.402
Optimal Principal Point [in pixels]	646.6	523.4	N/A

When comparing the undistorted errors from table 5.4 to that of the distorted errors in table 5.2, they show a significant decrease. The average centroid errors for each axis shows a 96.13% and 96.74% decrease respectively, and the distance to the centre errors shows a 98.65% decrease.

5.2.5 CubeStar Predistortion

For an uncalibrated star tracker to capture projected stars, correctly identify them and autonomously estimate the attitude, the stars need to be projected with the distortion of the camera applied before the star tracker captures the projected stars. The process of projecting the stars with distortion applied is referred to as "predistortion". Firstly an image I is generated, the stars in this image are then predistorted to result in image I_{pd} that is projected onto the monitor. When captured by the star tracker the lens distortion will cause a resulting image $I_{distorted}$, as the projected image is predistorted the lens distortion will cause the distorted image to be similar to the generated image such that $I_{distorted} = I$.

The predistortion model that is used in this project is the same radial distortion model used to undistort an image, however where the distortion model Q is used to undistort distorted star centroids such that $(x_u, y_u) = Q(x_d, y_d)$, to predistort centroids the distortion model is applied to the undistorted centroids such that $(x_{pd}, y_{pd}) = Q(x_u, y_u)$ where (x_{pd}, y_{pd}) are the predistorted centroid coordinates. Therefore, to predistort a generated image, for each star i the predistorted centroid is calculated with equations 5.29 and 5.30:

$$x_{pd}(i) = (x_u(i) - x_o) * (1 + K_1 * r_u(i)^2 + K_2 * r_u(i)^4 + K_3 * r_u(i)^6) + x_o \quad (5.29)$$

$$y_{pd}(i) = (y_u(i) - y_o) * (1 + K_4 * r_u(i)^2 + K_5 * r_u(i)^4 + K_6 * r_u(i)^6) + y_o \quad (5.30)$$

$$with\ r_u(i)^2 = (x_u(i) - x_o)^2 + (y_u(i) - y_o)^2$$

where

$x_{pd}(i), y_{pd}(i)$: predistorted coordinates whereto star i must be projected
$x_u(i)$, $y_u(i)$: undistorted (generated) coordinates of star i
r_u: distance from undistorted point to centre of distortion

The predistortion model in equations 5.29 and 5.30 was implemented in the software in the function "preDistort()", as shown in Algorithm 2, to predistort star centroids when it is requested by the user to do so. The results of the predistortion model are discussed in the next section.

5.2.6 CubeStar Predistortion Results

Figure 5.19 shows the generated calibration pattern with added predistortion that is projected onto the monitor, it can be seen that a predistorted calibration pattern has an inverted appearance than that of a distorted calibration pattern.

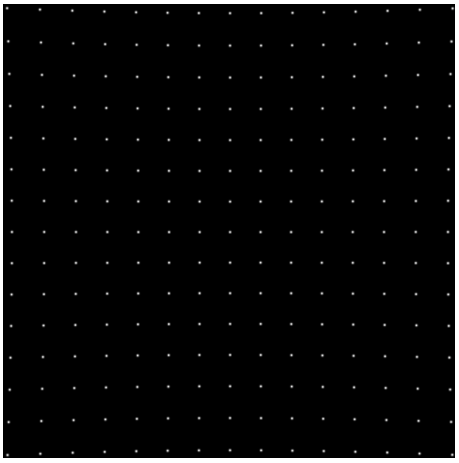


Figure 5.19: Generated Calibration Pattern With Predistorted Stars

Ideally the captured star centroids will be equal to the star centroids of a generated calibration pattern, however since the predistortion model used is the same as the radial distortion model the predistortion accuracies will be greatly dependant on the calibration results discussed in section 5.2.4. The measured calibration pattern that was captured by the star tracker of the predistorted calibration pattern in figure 5.19 is shown below in figure 5.20. A line from each corner star is added to the image to show that the stars are now aligned with one another and not in a pincushion distortion pattern, it is therefore clear that the measured image of a predistorted calibration pattern is much closer to a generated calibration pattern than the measured distorted calibration pattern in figure 5.15.

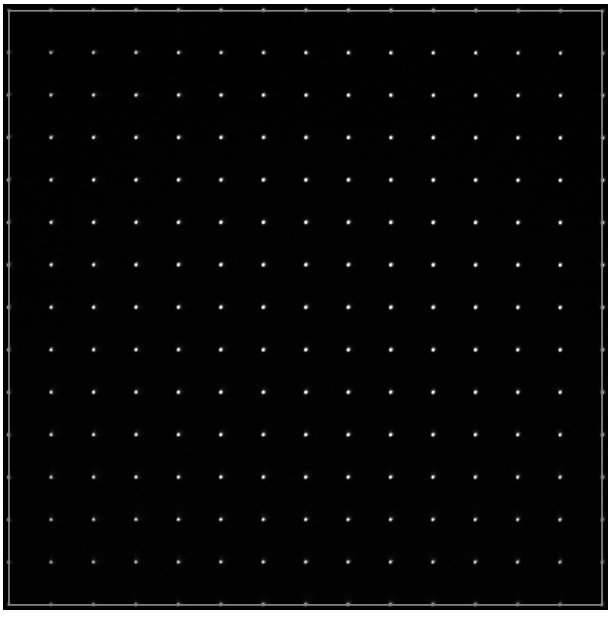
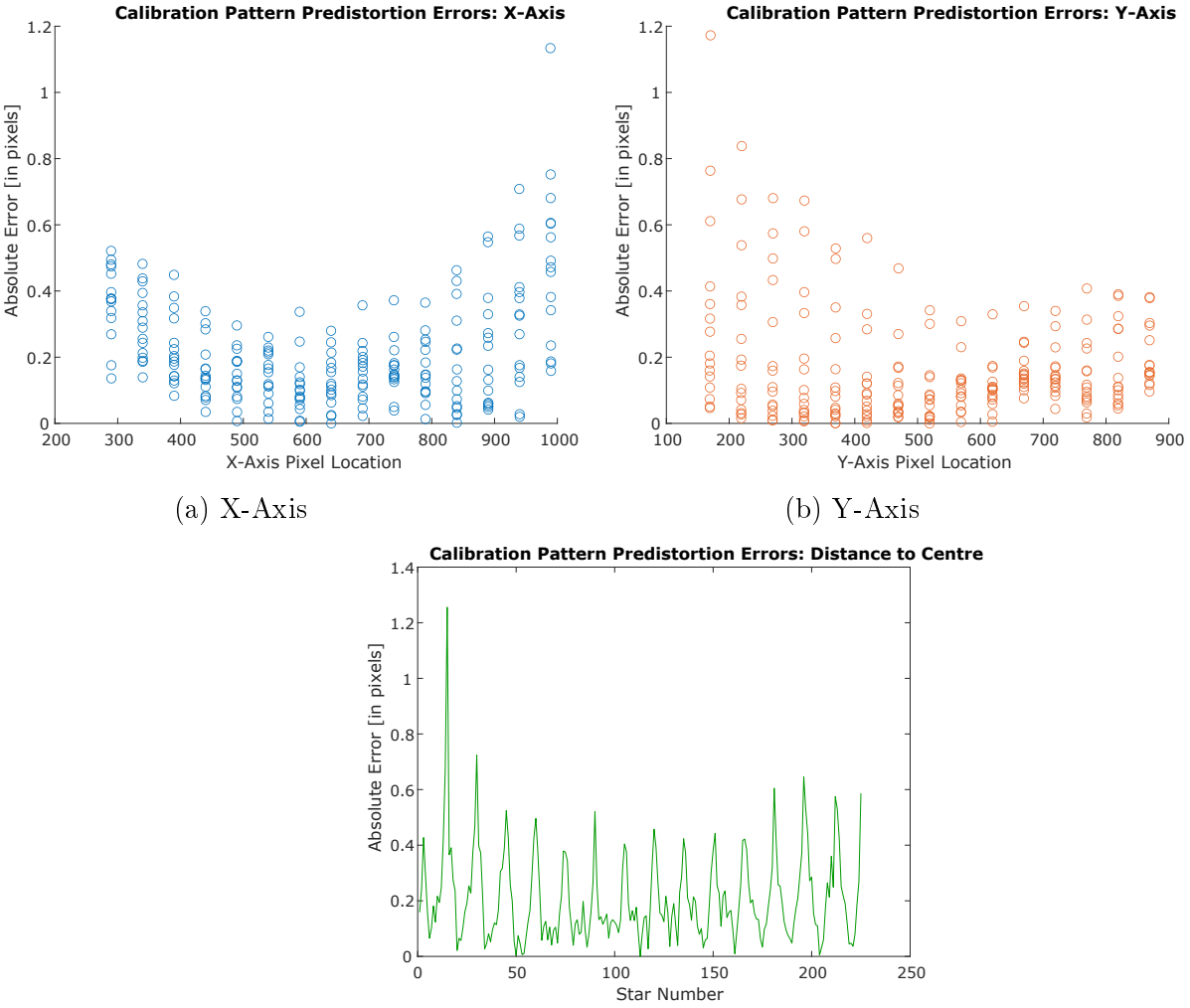


Figure 5.20: Captured Calibration Pattern Of Predistorted Stars

The accuracy of the predistortion model is evaluated by comparing the measured centroids of the measured predistorted calibration pattern in figure 5.20 to

the generated centroids in the generated calibration pattern with no added distortion in figure 5.14. The predistortion errors are shown in the figure 5.21 below. It is observed that while the errors are larger than the undistortion errors in figure 5.18, it is still a significant improvement on the distortion errors in figure 5.16.



(a) X-Axis

(b) Y-Axis

(c) Distance to Centre

Figure 5.21: Predistortion Errors Results

For the results in figure 5.21, the average absolute error for each axis and the average absolute distance error is summarized in table 5.5 below, along with the standard deviation for these errors and the maximum value of each error.

Table 5.5: Predistortion Error Results

	X-Axis	Y-Axis	Distance to Centre
Average Error [in pixels]	0.2161	0.1682	0.2074
Standard Deviation [in pixels]	0.1624	0.1691	0.1599
Maximum Error [in pixels]	1.1338	1.1725	1.2557

The results of the predistortion model are slightly higher than that of the undistortion model in table 5.4, therefore it is not as accurate. However, compared to the distorted errors in table 5.2 they still show a significant decrease in centroid errors. The average centroid errors for each axis shows a 93.72% and 95.26% decrease respectively, and the distance to the centre errors shows a 96.05% decrease. As predistortion of the star tracker lens in the emulation environment is a very complex problem, these values are accepted as the expected accuracy of the predistortion model used in this project.

5.3 Chapter Summary

The purpose of this chapter was to discuss the design and analysis of the emulation environment that satisfies the project objectives 2, 3, 4 and 5. The first section discussed the physical design aspects of the emulation environment and the components that were required. The emulation environment consist of a cardboard box used as a frame, a black cloth wrapped on the inside of the frame, the star tracker itself and its developer board, and a few components that were designed. These components consist of a stand for the CubeStar to be placed on that was built using plywood, a lens holder to hold the secondary lens that was 3D printed and a mount for the star tracker and lens holder that was also 3D printed. All of the components were placed inside the frame and the final emulation environment was shown in figure 5.10.

The second section discussed the analysis on the first practical data that was captured in the emulation environment. The results showed that three additional steps were required: an alignment correction, a calibration of the CubeStar and finally predistortion of the projected stars.

An alignment correction method was discussed that removes the vertical-, horizontal- and rotational offset between the CubeStar and the monitor by using the software to apply the required offsets to all projected stars. The alignment correction also provided a method to reduce the tilt- and swivel offsets to ensure that the monitor is parallel to the CubeStar's image plane. The results of the alignment correction showed that the vertical- and horizontal offsets were minimized to 0.02 pixels and 0.03 pixels respectively, and the rotational offset was reduced to 9.1 arcseconds. The tilt- and swivel offset were minimized to give resulting side length differences of 0.43 pixels for the horizontal sides and 0.61 pixels for the vertical sides.

The distortion correction procedure was discussed to determine the distortion coefficient of the radial distortion model, and the results showed that the star centroids could be undistorted with average errors of 0.1332 pixels on the x-axis and 0.1154 pixels on the y-axis. A predistortion method was then presented that projects stars to predistorted locations to enable the CubeStar to observe them at undistorted locations. The results of the predistortion showed that predistorted stars could be observed with average errors of 0.2161 pixels on the x-axis and

0.1682 pixels on the y-axis.

The results discussed in this section showed sufficient accuracies that would allow the CubeStar to estimate the attitude by capturing the projected stars. Therefore, the emulation performance of the CubeStar in the emulation environment could be measured to determine whether project objective 7 is satisfied.

Chapter 6

Emulation Performance Measurements

This chapter discusses the emulation performance measurements and results of the attitude estimation capabilities of the CubeStar inside the complete emulation environment to verify whether project objective 7 is satisfied. The results of the various aspects of the software and hardware discussed in chapters 4 and 5 showed that the errors caused by the emulation environment setup as well as the star tracker itself were minimised to an acceptable amount, therefore the final emulation performance of the star tracker can be evaluated. This chapter discusses the HIL experiments that are performed with the CubeStar placed inside of the emulation environment to capture stars projected onto the monitor.

The chapter is divided into three parts: 1) the attitude estimation accuracy with still images, 2) the angular rate estimation accuracy with stars projected at a constant angular velocity, and 3) the attitude estimation accuracy with stars projected at a constant angular velocity. An important note is that for all experiments that will be discussed in this chapter the maximum number of stars that is projected is limited to only 15 stars since the software program used to operate the CubeStar, known as "CubeSupport", has a built-in limit to only detect 15 stars. This limit is implemented to prevent the star detection algorithms onboard to exceed the allowed processing time. For locations where there are more than 15 stars in the FoV at the same time, the 15 stars closest to the boresight are projected.

6.1 Performance With Still Images

The first step to evaluate the performance of the CubeStar is to determine the attitude estimation accuracy with projected stars that are not moving. For this experiment 10 star images were projected onto the monitor to be captured by the CubeStar, each with a different boresight pointing star. To evaluate the performance of the star tracker, 5 different factors were investigated for each measurement:

1. The average distance of the projected stars to the boresight of the image.

2. The standard deviation of the distance of the projected stars to the boresight of the image.
3. The number of stars successfully identified.
4. The boresight angle errors.
5. The boresight rotation errors.

For each measurement the average distance of the projected stars to the boresight of the image and the standard deviation of this distance were obtained from the star projection software, the number of stars successfully identified was received from the CubeStar and the boresight angle errors and rotation errors were calculated from the generated attitude and the attitude quaternion estimate outputs of the CubeStar with the method discussed in section 3.3.4.

For each measurement the average distance and standard deviation of the projected stars to the boresight are used to visualize the spread of the projected stars across the FoV without the need to inspect the star image of each measurement. Visualizing the spread of the stars allows for more insight to the number of stars that were identified since stars further from the boresight is less likely to be identified due to higher distortion further from the boresight. Table 6.1 below summarizes the spread of the projected stars with four different cases based on the average and standard deviation of the distance to the boresight from each star.

Table 6.1: Spread Of Projected Stars Across FoV

Average Distance	Distance Standard Deviation	Resulting Star Spread
High	Low	Low spread, far from boresight. Most stars located far from the boresight.
High	High	Wide spread across entire FoV. Stars are scattered over the FoV.
Low	Low	Low spread, close to boresight. Most stars lie close to the boresight.
Low	High	Wider spread, but stars are still close to the boresight.

6.1.1 Attitude Estimation Accuracy

Since the attitude estimation accuracy is entirely dependant on the accuracy of the predistortion model discussed in section 5.2.5, for this experiment the 10 images were each projected and captured both with and without predistortion applied. To then verify the effectiveness of the predistortion model not only the angle- and

rotation errors were inspected but also the number of stars that were detected and successfully identified by the CubeStar, as this is an effective method to verify that the predistortion model is effective. The number of stars that was detected and successfully identified will first be discussed, followed by the angle- and rotation errors.

Table 6.2 below summarizes the amount of stars that were detected for each measurement as well as the amount of stars that were successfully identified for measurements both with and without predistortion applied. The average and standard deviation of the distance to the boresight from the projected stars are also included in the table to aid the discussion that follows. The results in the table are sorted in increasing order for the average distance.

Table 6.2: Number Of Stars Identified By CubeStar

		Number Of Stars Identified		Distance To Boresight From Stars	
Boresight Star ID	Number Of Stars Detected	No Predistortion	With Predistortion	Average [in pixels]	Standard Deviation [in pixels]
45556	15	14	15	125.30	65.11
26311	15	12	14	129.41	82.28
87073	15	13	15	132.15	72.80
60718	15	10	15	144.79	80.00
69996	15	9	15	163.92	80.95
34444	15	7	15	167.33	101.18
76600	15	7	15	185.19	85.22
4427	15	0	14	237.02	154.95
19949	15	0	13	270.67	66.30
84500	15	0	13	297.11	89.55

From the second column in table 6.2 it can be seen that the selected boresight stars all resulted in star images where at least 15 stars were in the FoV at the same time. This was intentional to provide a consistent number of detected stars, however, the boresight stars were chosen to result in a variety of different star spread across the FoV for each star image, as shown in the last two columns.

The results show that for projected stars that lie very close to the boresight the CubeStar is able to identify a surprisingly high amount of stars with distortion present, as shown by the top 3 entries in the table 6.2. In those three measurements all of the stars were located very close to the boresight as indicated by the low average distance and standard deviation shown in the last two columns. This again proves that the distortion close to the boresight is extremely low, however the lower entries show that as soon as the stars are located further from the boresight the number of identified stars drastically decreases, which indicates that the distortion further from the boresight increases to the point where the angular distances of the stars are too distorted for the stars to be identified. The last two

entries is a perfect example, a much higher average distance than the top entries and a very low standard deviation indicates that most of the stars are located very far from the boresight, therefore the CubeStar could not identify any stars with no predistortion applied.

The effectiveness of the predistortion model is proven by the fourth column in the table, which shows a dramatic increase in the amount of identified stars for all measurements of predistorted stars. In most cases all of the projected stars that were predistorted were detected, even the last two entries where zero distorted stars could be identified showed an increase of 13 identified stars with predistortion applied. The results in table 6.2 therefore show that the predistortion model is very effective, however the last two entries also show that the accuracy of the predistortion model has its limitations due to the fact that for each of the two entries there were two stars located very far from the boresight, showing that the predistortion model is not 100% effective at locations very far from the boresight.

The attitude estimation accuracy could now be evaluated by determining the boresight angle- and rotation errors of the 10 predistorted star images. The results are shown in figure 6.1 below. The average distance and standard deviation for each measurement is also displayed on the figure, with its own secondary y-axis indicated on the right of the figure. The measurements are also ordered in increasing average distance.

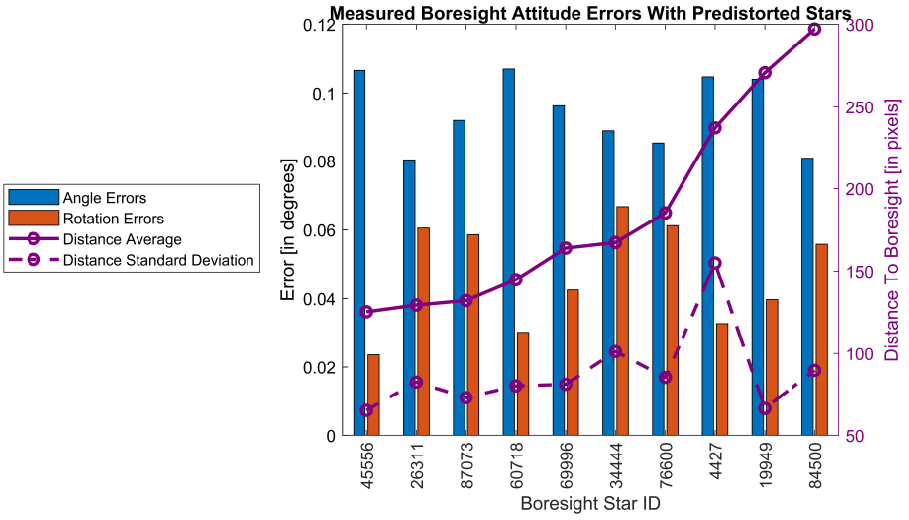


Figure 6.1: Measured Attitude Errors With Predistorted Stars

Figure 6.1 shows promising results as the maximum angle error was only 0.107° and the maximum rotation error 0.0667°. The average angle error was 0.0946° and the average rotation error was 0.0472°. While lower errors would certainly be preferred, the achieved results are as expected based on the shown effectiveness of the predistortion model in section 5.2.6 for the current emulation environment setup. One would expect projected star images with a wider spread of stars to deliver larger angle errors with lower rotation errors and that projected star images

with stars much closer to the boresight would show lower angle errors with higher rotation errors. These expectations, however, are not present in the results shown in figure 6.1. Although the errors are consistently as low as expected, they do not show a consistent pattern between the error results compared to the spread of the stars across the FoV.

6.1.2 Conclusion

Based on the results discussed in this section, the results of the alignment correction in section 5.2.2 and the results of the predistortion model in section 5.2.6, conclusions were made given that the following criteria are met:

1. The CubeStar used inside the emulation environment must be positioned with a centre star centroid within 0.1 pixels of the centre of the image, based on the results achieved in section 5.2.2.

2. The tilt- and swivel alignment between the CubeStar's image plane and the monitor must be accurate enough to result in the side length errors of the alignment pattern to be less than $1 pixel$, based on the results achieved in section 5.2.2.

If the above criteria are met, the results in table 6.2 show that all stars within a window of approximately 700×700 pixels around the boresight would be successfully identified. This window is referred to as the star identification window, although stars outside of the star identification window could still be successfully identified, the results in table 6.2 show that it is less likely. The results in figure 6.1 further shows that if the above criteria are met and there are stars located inside of the star identification window, the CubeStar could achieve angle errors below approximately 0.107° and rotation errors below approximately 0.0667°.

6.2 Performance With Constant Angular Rates

The next important performance measurements are the angular rate estimation and attitude estimation of the CubeStar when the stars are projected with constant angular rates that emulate movement of the star tracker. The CubeSupport software is used to enable tracking mode on the CubeStar and start a logging sequence. The logging feature sends a trigger command to the CubeStar to request an attitude quaternion estimate at a frequency of 1Hz, therefore the CubeStar captures the projected stars at a sampling rate of 1Hz. The resulting log file then contains the following relevant data for each measurement: the current timestamp in hours, minutes, seconds and milliseconds, the number of stars detected and identified, the attitude quaternion estimate and lastly the estimated angular rate about the x-, y- and z-axis. To measure the accuracy of the angular rate estimation and the attitude estimation with moving stars, the software that projects the stars also creates a log that contains the information for every star image that is generated and projected for each updated attitude. The data that is logged by the star projection software includes the following: the current timestamp in

hours, minutes, seconds and milliseconds, the generated attitude quaternion, the average distance to the boresight from each star and the standard deviation of this average distance. The log entries from CubeSupport can then be compared to the log entries from the star projection software to find the angular rate- and attitude estimation errors.

After initial experiments it was observed that CubeSupport and the star projection software was not synchronized. The result was that projected stars were updated before they were captured by the CubeStar which in turn caused offsets in the logged data that would affect the results. The solution to ensure synchronization between the two software programs is discussed in this section, after which the angular rate estimation accuracy will be discussed and lastly the attitude estimation accuracy.

6.2.1 Synchronization Of Star Projection and Capture

The most optimal way to ensure synchronisation between the two software programs would be to not use CubeSupport at all and instead adapt the existing star projection software to communicate with the CubeStar directly. However, since this would be a very time consuming process, a simpler and effective alternative method was explored to ensure synchronization between the projection of the stars and the star tracker capturing them. After some tests and an investigation of the timing diagram for the sequence of events of the CubeStar, a timing diagram for the star projection software was created to plan the sequence of events to prevent projected star images from being captured twice or not captured at all, therefore ensuring synchronization between CubeSupport and the star projection software.

The first step was to measure the starting time delay, t_{start}, from when the log is started from CubeSupport until the first log entry's timestamp. The results showed an average delay of 145.44ms (from 10 measurement sets) and a maximum delay of 175ms. The starting time delay is therefore set to a default time of $t_{start} = 200$ms and will be implemented in the timing diagram. For any possible additional delays, the timing diagram of the CubeStar was investigated in the reference manual [18]. The reference manual only showed one additional delay that would be of concern, a trigger delay t_{trig}. For the CubeStar to achieve an attitude quaternion estimate output frequency of 1Hz it captures a new image while the previous image is being processed, to ensure that each image receives enough processing time a default trigger delay of $t_{trig} = 500$ms after a request is received is used before a new image is captured.

The final timing diagram is shown in figure 6.2 below, it shows the sequence of events with a total of 4 timing delays taken into consideration: the starting time delay t_{start} and the trigger delay t_{trig} that were both discussed above, an exposure time of $t_{exp} = 18$ms which indicates the time that the CubeStar will take to capture an image and the last delay is the update delay t_{update} which is the time that the projection software waits after each time step tick before it updates

the projected star image. After the projection is started at time $t0$, the timing diagram shows a time step tick at a frequency of 1Hz.

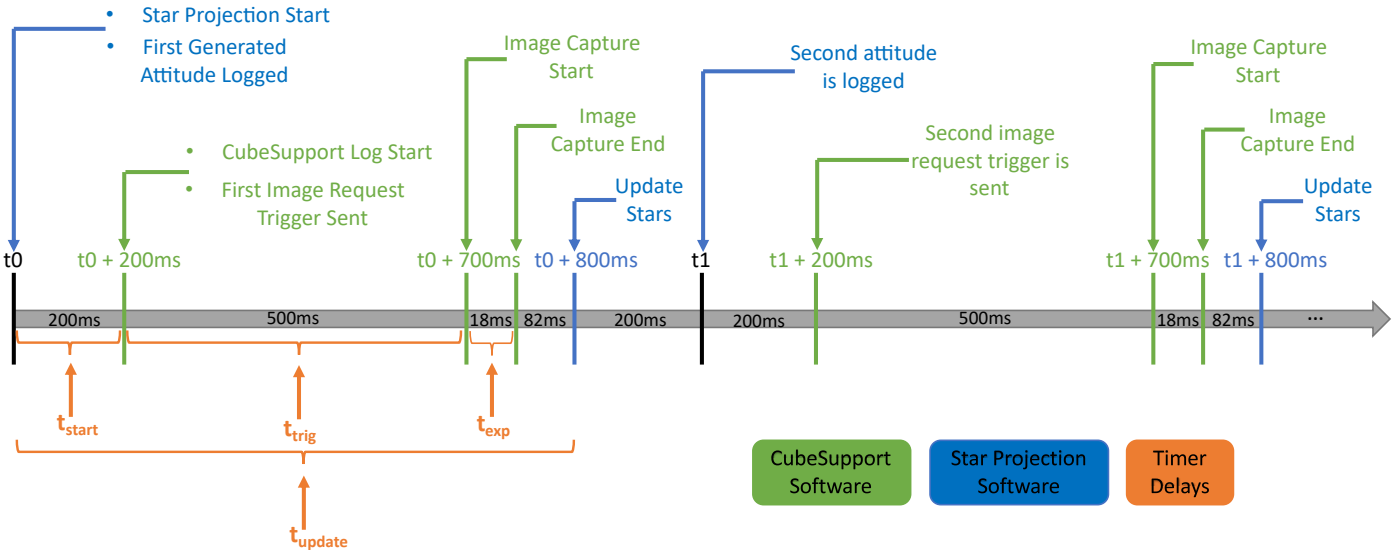


Figure 6.2: Timing Diagram: Sequence of events for CubeSupport and star projection software

The timing diagram show that even if the events of CubeSupport are delayed with the maximum starting time delay of 200ms, the update delay is set to $t_{update} = 800$ms to prevent the projected stars from being updated too late or too early. With the use of the accurate software timer discussed in section 4.3.2, a timer with a refresh rate of 10Hz is used to check the elapsed time every 100ms, as soon as the elapsed time reaches the correct value the projected star image is updated.

The timing diagram discussed in this section ensures that each entry in the output log from CubeSupport matches with its counterpart in the star projection software's output log, irrespective of the timestamps of the logs. The timing diagram proven to be very effective as the analysed data showed very accurate rate estimation and no missed measurements, as will be discussed in the next section.

6.2.2 Angular Rate Estimation Accuracy

To determine the angular rate estimation accuracy of the CubeStar in the emulation environment emulations were performed where the projected stars are moved at a constant angular rate with the method discussed in section 4.3. The output logs from CubeSupport containing the measured data was then analysed to find the angular rate estimation accuracy for different constant angular rates. The emulations performed consist of multiple sets of star projections with different constant angular rates for each axis. Each set consist of three subsets where the respective angular rate for that set was first applied to only the x-axis, then to both the x- and y-axes and lastly to all three axes. Each subset had a runtime of

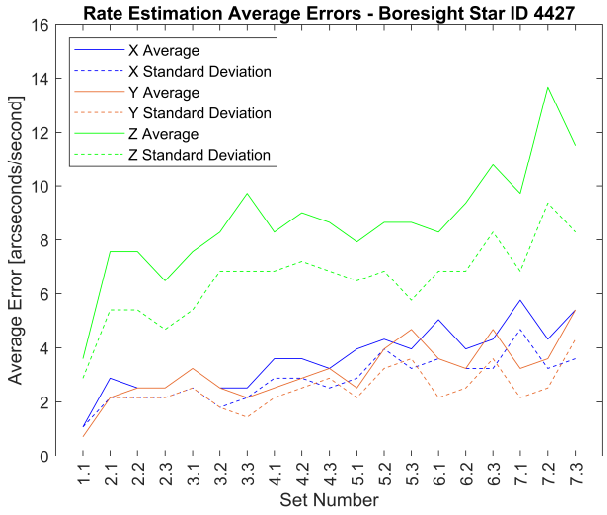
60 seconds to result in 60 measurements per subset that could be analysed. The projected constant angular rates for each subset are shown in table 6.3 below.

Table 6.3: Projected Angular Rates

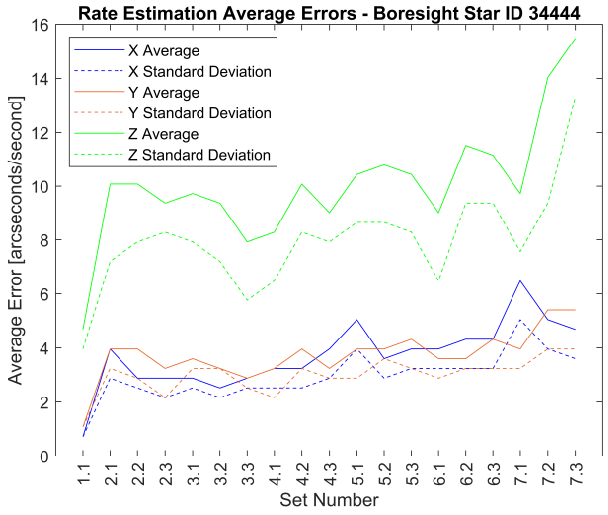
	Set Number	1.1	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3
Angular Rate [in °/s]	X-Axis	0	0.05	0.05	0.05	0.1	0.1	0.1	0.15	0.15	0.15
	Y-Axis	0	0	0.05	0.05	0	0.1	0.1	0	0.15	0.15
	Z-Axis	0	0	0	0.05	0	0	0.1	0	0	0.15

	Set Number	5.1	5.2	5.3	6.1	6.2	6.3	7.1	7.2	7.3
Angular Rate [in °/s]	X-Axis	0.2	0.2	0.2	0.25	0.25	0.25	0.4	0.4	0.4
	Y-Axis	0	0.2	0.2	0	0.25	0.25	0	0.4	0.4
	Z-Axis	0	0	0.2	0	0	0.25	0	0	0.4

The set of angular rates were applied to three different sets of emulations, each with a different starting boresight pointing star, to cover a wider range of boresight pointing locations. Even though the starting boresight pointing star will only be on the boresight for the first second of the emulation, the star ID's are used to represent the three emulation sets in the results. To find the angular rate estimation errors, the output logs of the entire data set was analysed. For each log file, the estimated rate of each axis for all of the entries were compared to the generated angular rate for each respective axis to find the angular rate estimation errors. The average rate estimation error of each axis was calculated for each subset's log file, as well as the standard deviation of the angular rate estimation error. The results of the entire data set are illustrated in figure 6.3 below. Each figure shows the average rate estimation error and standard deviation on each axis, for each subset.



(a) Boresight Star ID 4427



(b) Boresight Star ID 34444

(c) Boresight Star ID 60718

Figure 6.3: Rate Estimation Accuracy

Figures 6.3a to 6.3c show that the CubeStar can estimate the angular rates with extremely high accuracies. For the x- and y-axes the highest average error across the entire data set was only 6.48″/s and 5.76″/s respectively. The figures show that the z-axis consistently delivers higher errors than the other two axes, however the highest error across the entire data set is still only 15.48″/s.

The angular rate estimation results therefore show that the angular rate projection method discussed in section 4.3 is very effective and that the timing diagram discussed in section 6.2.1 to synchronise the star projection software with CubeSupport is also effective. Therefore the attitude estimation accuracy of the CubeStar with stars projected with constant angular rates could now be determined.

6.2.3 Attitude Estimation Accuracy

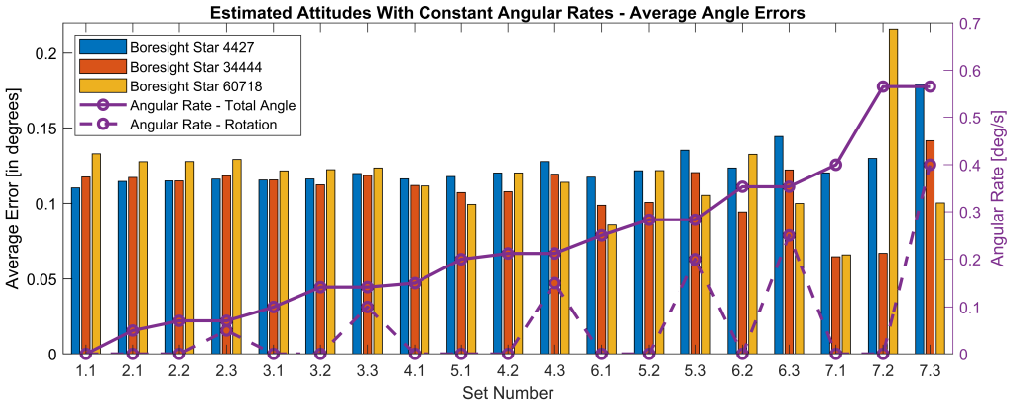
The attitude estimation accuracy of the CubeStar in the emulation environment is the last performance measurement that must be discussed to determine the effectiveness of the emulation environment. As mentioned in the previous section, the angular rate estimation results showed that the timing diagram discussed in section 6.2.1 successfully synchronises the star projection software with CubeSupport. Furthermore, the angular rate estimation results analysis in the previous section showed no log entries where the CubeStar captured the same projected star image twice or missed one projected star image, which gives even more confidence in the timing diagram. Therefore, the attitude estimation accuracy could now be determined by analysing the output logs of the same data set that was used in the previous section. To find the attitude estimation error of each subset the attitude quaternion estimate of each measurement in the respective CubeSupport output log file is compared to its generated attitude quaternion counterpart in the output log file from the star projection software to calculate the angle- and rotation error for each of the 60 measurements with the method discussed in section 3.3.4. The average angle- and rotation error and their standard deviation is then calculated for each subset.

To summarize the results of the entire data set, the average angle- and rotation errors and standard deviations of each subset are shown in figures 6.4a to 6.4d. Each figure also show the angular rates of each subset represented as the total angle about the x- and y-axis combined and the rotation about the z-axis. The results are then ordered firstly by increasing total angle and secondly by increasing rotation.

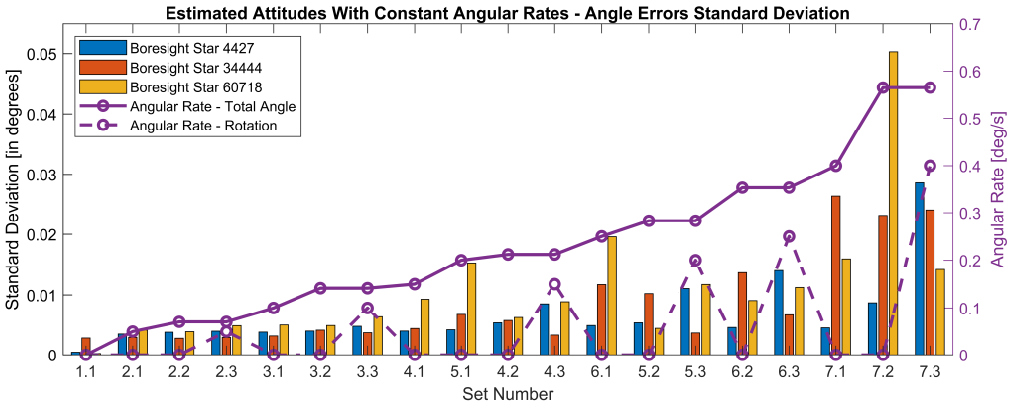
For the angle errors, figure 6.4a show that the average angle errors remain fairly constant across all of the sets, however figure 6.4b shows that the standard deviations increases as the angular rate increases. The results show that for angular rates up to 0.25°/s on each axis the average angle errors remained below 0.145° and the standard deviations remained below 0.016° (with the exception of one outlier above this value). For angular rates above 0.25°/s the average angle errors become more inconsistent, as well as their standard deviations.

For the rotation errors, figures 6.4c and 6.4d show that both the average rotation errors and standard deviations increases as the angular rates increases. The results show that for angular rates up to 0.25°/s on each axis the average rotation errors remained below 0.1° (with the exception of two outliers above this value) and the standard deviations remained below 0.031° (also with only two outliers above this value). For angular rates above 0.25°/s the average rotation errors and standard deviations for emulation sets 34444 and 60718 increased dramatically.

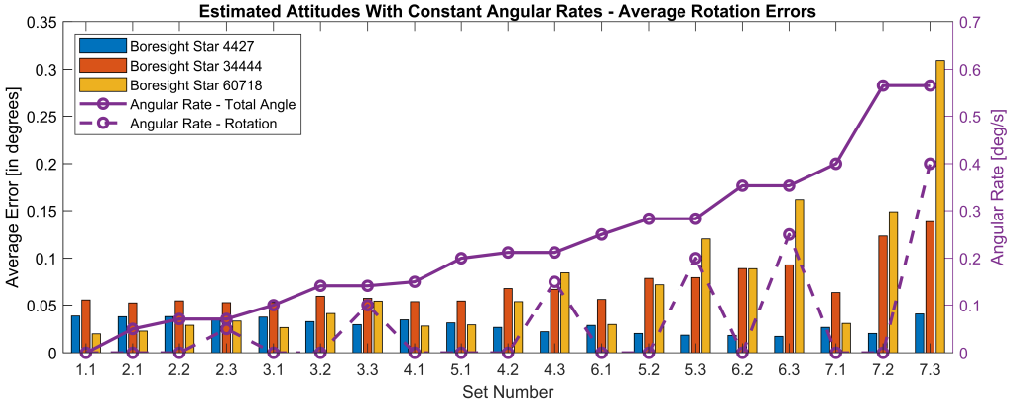
The larger angle- and rotation errors at higher angular rates are caused by a combination of two factors: 1) the spread of the stars and 2) the effectiveness of the predistortion model. With higher angular rates the boresight reached locations where the average distance to the boresight from each star in the FoV is very



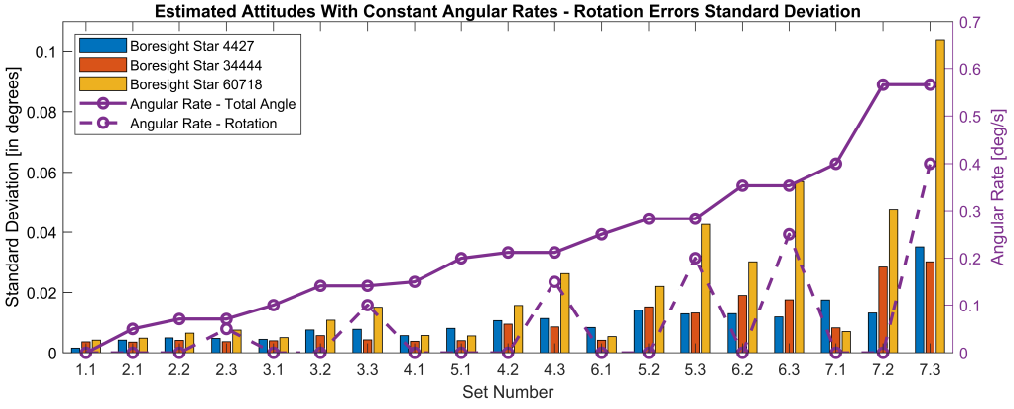
(a) Average Angle Errors



(b) Angle Errors Standard Deviation



(c) Average Rotation Errors



(d) Rotation Errors Standard Deviation

Figure 6.4: Attitude Estimation Accuracy With Constant Angular Rates

large, which indicates that the stars are located far from the boresight where the predistortion model is much less effective, as mentioned in section 6.1 in the discussion of the results of table 6.2. As the stars move across the FoV a lot of the projected stars move outside of the star identification window mentioned in section 6.1.2, where the predistortion model is proven to be ineffective. Therefore, when the boresight reaches a location where no stars are located inside of the star identification window it is likely that the attitude estimation errors will become very large.

6.3 Chapter Summary

The purpose of this chapter was to determine whether project objective 7 was satisfied. The final performance measurements of the CubeStar in the emulation environment was evaluated with various HIL experiments, these experiments consisted of the CubeStar capturing both still projected stars and stars that were projected with constant angular rates. The purpose of the experiments were to evaluate the attitude- and rate estimation capabilities of the CubeStar in the emulation environment.

The results of the still images in the first section showed that firstly the predistortion model proved very effective for stars that are located within a window of 700×700 pixels around the boresight and that the CubeStar could achieve angle errors below 0.107° and rotation errors below 0.0667° given that there are stars located in this window.

To evaluate the performance of the CubeStar during experiments with stars projected at constant angular rates, a timing diagram to plan the sequence of events for the star projection software and CubeSupport was first discussed, the purpose of the timing diagram was to ensure synchronisation between the two software programs to ensure that the CubeStar captures every projected star image without missing a single image or capturing the same image multiple times.

The first performance measurement with constant angular rates that was discussed was the angular rate estimation accuracy. The experiments performed included three emulation sets with different starting boresight pointing stars, each with 19 subset with different angular rates for each axis. The CubeStar showed excellent rate estimation accuracies with maximum rate estimation errors of $6.48''/s$ in the x-axis, $5.76''/s$ in the y-axis and $15.48''/s$ in the z-axis.

The last performance measurement that was discussed was the attitude estimation accuracy with constant angular rates. The results showed that the CubeStar was capable of tracking the attitude with an angular rate of up to $0.25^\circ/s$ on each axis with resulting angular errors below 0.145° and rotation errors below 0.1°.

Chapter 7

Conclusions And Recommendations

This chapter gives a summary of the entire project to show that all the project objectives were achieved and therefore it can be concluded that the project is a success. Finally, there is a brief discussion on recommendations and future work that could lead to improvements on the emulation environment that was created in this project.

7.1 Summary and Conclusion

Since each chapter is concluded with its own summary, this final summary is focussed on how the project objectives were achieved by discussing three aspects: the star projection software, the emulation environment and the emulation performance measurements.

Star Projection Software:
After star image generation and processing algorithms were investigated in chapter 3, a software program was written that can project stars onto a monitor to be captured by the CubeStar.

The star projection software implements the following aspects to satisfy project objective 1: a star vector list that contains the inertial vectors of the stars, a TRIAD estimator to generate an initial attitude, an inertial vector to image plane coordinates conversion to find the star centroids, and finally a Gaussian distribution to generate a unique star profile for each star based on their centroid location to accurately represent a real night sky star. To establish how effective the star projection software is and to verify whether project objective 1 was satisfied, star detection and identification algorithms were used to determine the accuracy to which star images were generated. The star detection algorithms uses the Image Plane Search-, Region Growing- and Centroiding algorithms to find the centroid of the projected stars and the GVA identifies the stars. The QUEST algorithm then calculates the estimated attitude using the inertial vectors from the star catalogue of the identified stars. With an analysis of generated star images it was determined that the stars are generated with average centroid errors of 0.0181 pixels in the x-axis and 0.0185 pixels in the y-axis. The attitudes could be generated with

an average angle error of 2.47 arcseconds and an average rotation error of 1.45 arcseconds, therefore project objective 1 was satisfied.

To satisfy project objective 6 the stars are projected with constant angular rates by using kinematic equations that were investigated to update the star tracker's attitude at regular intervals. When the attitude is updated with the constant angular rate and the correct sampling time, the new star locations are calculated and the stars are projected to these new locations. To ensure that the stars are updated at constant intervals, an accurate software timer was investigated. The software timer showed an average error close to zero and is therefore capable of updating the star tracker's attitude at an accurate interval to ensure that the stars are projected with constant angular rates and therefore project objective 6 was satisfied.

Emulation Environment:
An emulation environment was built for this project that contains a LCD monitor on which stars are projected to be captured by the star tracker. A frame was constructed using cardboard boxes and the inside was wrapped with black cloth, the cardboard frame blocked out most of the light and the black cloth blocked out any light that may penetrate the frame, however the main purpose of the cloth was to prevent most of the light refracting from the monitor. A plywood stand was designed and built on which the star tracker could be placed to capture the projected stars on the monitor. With all of the components placed in the emulation environment, project objective 2 was satisfied. The size of the entire emulation environment was $0.8\text{m} \times 0.6\text{m} \times 0.5\text{m}$, which satisfied project objective 3. A LCD monitor that was already available in the ESL was used to minimize cost and the rest of the components that were used were very inexpensive, therefore project objective 4 was satisfied.

For the emulation environment to be suitable for a calibrated or uncalibrated CubeStar to satisfy project objective 5, further adjustments had to be made to the environment. The lens of an uncalibrated CubeStar can be focussed on the monitor, however a calibrated CubeStar will have a lens focussed on infinity that cannot be adjusted, therefore a secondary lens was used that allows a CubeStar focussed on infinity that is placed at a distance of 333.33mm from the monitor to be able to focus on the monitor inside the emulation environment. Furthermore, various steps were taken to ensure that the CubeStar with the secondary lens were aligned with the monitor and that distortion errors were reduced. Firstly, an alignment correction procedure was presented which showed that the CubeStar could be aligned to the monitor with the alignment offset errors minimized to the values shown in table 7.1 below.

Table 7.1: Alignment Correction Errors

Offset	Error Value
Vertical	0.02 pixels
Horizontal	0.03 pixels
Rotational	9.1 arcseconds
Tilt	0.43 pixels
Swivel	0.61 pixels

Secondly, a calibration procedure was presented to find the distortion coefficients of an even-order polynomial radial distortion model that was investigated. With the CubeStar aligned to the monitor, the measured distorted star centroids were compared to the generated star centroids and the results showed average star centroid errors of 3.4427 pixels in the x-axis and 3.5448 pixels in the y-axis, which indicated that distortion correction was vital. To determine how effective an image could be undistorted with the radial distortion model after the coefficients were determined with the calibration procedure, measured star centroids were undistorted and the undistorted star centroids were compared to the generated star centroids, the results showed that the average undistorted star centroid errors were minimized to 0.1332 pixels in the x-axis and 0.1154 pixels in the y-axis.

Lastly, a predistortion method was presented that could compensate for the radial distortion that is present by projecting the stars to predistorted locations. The predistorted stars allows the CubeStar to observe them at undistorted locations, this allows the CubeStar to identify the stars and estimate the attitude autonomously. The predistortion model that was used was the same radial distortion model used to undistort an image, however the predistortion model is applied to distortion-free generated star images to produce predistorted star images. To determine how effective the predistortion model was, predistorted stars were captured and their centroids were compared to the distortion-free generated centroids. The results showed that predistorted stars could be observed with average errors minimized to 0.2161 pixels on the x-axis (93.72% decrease compared to distorted centroids) and 0.1682 pixels on the y-axis (95.26% decrease compared to distorted centroids).

Emulation Performance Measurements:
The final emulation performance of the CubeStar in the emulation environment was measured to determine whether project objective 7 was satisfied, and therefore how successful the project was. To determine the final emulation performance measurements various HIL experiments were performed with both still projected stars and stars that were projected with constant angular rates to determine the attitude estimation- and rate estimation accuracies. For still projected stars the CubeStar showed angle errors below 0.107° and rotation errors below 0.0667° given that there were stars located in a star identification window of 700×700 pixels around the boresight. For stars that were projected with constant angular rates, the CubeStar showed average angle errors below 0.145° and average rotation er-

rors below 0.1° for angular rates up to 0.25°/s on each axis. The maximum rate estimation errors of the CubeStar resulted in 6.48″/s in the x-axis, 5.76″/s in the y-axis and 15.48″/s in the z-axis.

In conclusion, with all the project objectives satisfied, the project was deemed a success. The results of the emulation performance measurements show that the emulation environment is suitable for an uncalibrated CubeStar to capture projected stars and identify them, even when the stars are projected with a constant angular rate. Therefore, when modifications to the algorithms onboard a CubeStar is made, they can be tested with HIL experiments inside the emulation environment to determine whether the modifications produce better results than older versions of the algorithms, without the need to capture the night sky. The emulation environment would not be suitable to determine the final performance measurements of a star tracker, however there are some recommendations for improvements that could be made to possibly produce even better results than that of this project, these recommendations are discussed in the next section.

7.2 Recommendations and Future Work

This section presents improvements that could be made to the emulation environment that was designed and built for this project. The improvements that are presented has the potential of improving the results that were achieved.

OLED Monitor

The first recommendation is to replace the LCD monitor with an OLED monitor. The LCD monitor used in this project has an edge-lit LED backlight that is used to pass light through a pixel. Since the backlight is edge-lit at the bottom of the monitor (like most commercial LCD monitors), pixels at the bottom of the monitor appear much brighter than pixels at the top, this causes stars projected at the bottom of the monitor to have brighter magnitudes than stars projected to the top. OLED monitors have pixels that each produce their own light, which results in a more even spread of light and therefore the magnitude of the stars will be the same at each location on the monitor. Another benefit of pixels that produce their own light is that when a pixel is switched off, there is no light emitted from that pixel, compared to a LED backlight that remains on even when there are no pixels that are switched on. This benefit would result in less light that is reflected inside the emulation environment from pixels that are not switched on, which will potentially lead to stars being observed with better accuracy.

Improved Stand For Star Tracker

For this project, a plywood stand was built on which the CubeStar could be placed. This stand was independent of the monitor and therefore could accidentally be shifted to another position, which causes a misalignment between the CubeStar and the monitor. It was also noted that the position of the CubeStar shifted over the duration of a couple of hours, this could be due to the plywood warping due to the heat on the inside of the environment. With room temperature outside of the

environment, the inside reached temperatures of over 35°C. Although the alignment correction procedure presented in this thesis was effective, it was not very efficient as it was very time consuming. Since an alignment correction procedure is required with even a slight movement of the CubeStar, it is recommended that an alternative platform is designed and built for future work. Firstly, the platform should either provide a mounting location for the monitor onto the platform or provide an effective way to secure the platform to the monitor. This will prevent any large movements between the CubeStar and the monitor and would decrease the presence of tilt- and swivel offsets between the two. Secondly, the platform on which the star tracker is placed must be designed to allow for much finer 3-axis adjustments. This will speed up the alignment correction procedure significantly. An alternative material than plywood could be explored to produce parts that could enable these finer adjustments.

Project Star Magnitudes

For this project the magnitude of each star from the catalogue was not considered when generating their star profile. This decision was made due to the edge-lit LED backlight of the monitor that was used. The backlight already introduces variations in star magnitudes across the monitor due to it being edge-lit at the bottom of the monitor, it was therefore decided that projecting stars with varying magnitudes would result in too many inconsistencies to measure the CubeStar's performance reliably. When an OLED monitor is used, it is recommended to take the visual magnitude of each star into consideration when generating the star profile to more accurately simulate the stars in the night sky. If an OLED monitor is not used and instead a LCD monitor such as the one used in this project, the distribution of the light on the monitor could be measured and a method to apply this distribution to the magnitude of the stars could be investigated.

List of References

[1] Nasa's latest smartphone satellite ready for launch. 2014. Accessed: 20-01-2021. Available at: `https://www.nasa.gov/centers/ames/engineering/projects/phonesat2.html`

[2] Kule, E.: Nanosats database. 2021. Accessed: 20-01-2021. Available at: `https://www.nanosats.eu/#figures`

[3] Heidt, H., Puig-Suari, J., Moore, A., Nakasuka, S. and Twiggs, R.: Cubesat: A new generation of picosatellite for education and industry low-cost space experimentation. 2000.

[4] Puig-Suari, J., Turner, C. and Ahlgren, W.: Development of the standard cubesat deployer and a cubesat class picosatellite. In: *2001 IEEE aerospace conference proceedings (cat. No. 01TH8542)*, vol. 1, pp. 1–347. IEEE, 2001.

[5] Poghosyan, A. and Golkar, A.: Cubesat evolution: Analyzing cubesat capabilities for conducting science missions. *Progress in Aerospace Sciences*, vol. 88, pp. 59–83, 2017.

[6] Erlank, A.O.: *Development of CubeStar A CubeSat-Compatible Star Tracker*. Master's thesis, Stellenbosch University, 2015.

[7] CubeSpace: Cubestar star tracker. Accessed: 20-01-2021. Available at: `https://www.cubespace.co.za/`

[8] *CubeStar Interface Control Document*. CubeSpace, .

[9] Perryman, M. *et al.*: The hipparcos and tycho catalogues. *star*, vol. 2, p. 2π1, 1997.

[10] Liebe, C.C.: Accuracy performance of star trackers-a tutorial. *IEEE Transactions on aerospace and electronic systems*, vol. 38, no. 2, pp. 587–599, 2002.

[11] Rufino, G., Accardo, D., Grassi, M., Fasano, G., Renga, A. and Tancredi, U.: Real-time hardware-in-the-loop tests of star tracker algorithms. *International Journal of Aerospace Engineering*, vol. 2013, 2013.

[12] Roshanian, J., Yazdani, S. and Ebrahimi, M.: Star identification based on euclidean distance transform, voronoi tessellation, and k-nearest neighbor classification. *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 6, pp. 2940–2949, 2016.

[13] Samaan, M.A., Steffes, S.R. and Theil, S.: Star tracker real-time hardware in the loop testing using optical star simulator. *Spaceflight Mechanics*, vol. 140, 2011.

[14] Herman, L.: The history, definition and pecularities of the earth centered inertial (eci) coordinate frame and the scales that measure time. In: *1995 IEEE Aerospace Applications Conference. Proceedings*, vol. 2, pp. 233–263. IEEE, 1995.

[15] Diebel, J.: Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[16] Trawny, N. and Roumeliotis, S.I.: Indirect kalman filter for 3d attitude estimation. *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep*, vol. 2, p. 2005, 2005.

[17] Vizier. Accessed: 01-02-2020.
Available at: `https://vizier.u-strasbg.fr/viz-bin/VizieR`

[18] *CubeStar Reference Manual*. CubeSpace, .

[19] Shuster, M.D.: The optimization of triad. *The Journal of the Astronautical Sciences*, vol. 55, no. 2, pp. 245–257, 2007.

[20] Greyling, B.C.: *A Charge Coupled Device Star Sensor System for a Low Earth Orbit Microsatellite*. Master's thesis, Stellenbosch University, 1995.

[21] Bukhari, F. and Dailey, M.N.: Automatic radial distortion estimation from a single image. *Journal of mathematical imaging and vision*, vol. 45, no. 1, pp. 31–45, 2013.

[22] Kukelova, Z. and Pajdla, T.: A minimal solution to radial distortion autocalibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 12, pp. 2410–2422, 2011.

[23] Roux, G.J.: *Augmented Stellar Sensor for a Small Spacecraft*. Master's thesis, Stellenbosch University, 2019.

[24] Brown, D.C.: Decentering distortion of lenses - the prism effect encountered in metric cameras can be overcome through analytic calibration. *Photometric Engineering*, vol. 32, no. 3, pp. 444–462, 1966.

[25] Kolomenkin, M., Pollak, S., Shimshoni, I. and Lindenbaum, M.: Geometric voting algorithm for star trackers. *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 2, pp. 441–456, 2008.

[26] Wahba, G.: A least squares estimate of satellite attitude. *SIAM review*, vol. 7, no. 3, pp. 409–409, 1965.

[27] Markley, F.L.: Attitude determination using vector observations and the singular value decomposition. *Journal of the Astronautical Sciences*, vol. 36, no. 3, pp. 245–258, 1988.

[28] Markley, F.L. and Mortari, D.: How to estimate attitude from vector observations. 1999.

[29] Shuster, M.D.: The quest for better attitudes. *The Journal of the Astronautical Sciences*, vol. 54, no. 3-4, pp. 657–683, 2006.

[30] Shuster, M.D. *et al.*: A survey of attitude representations. *Navigation*, vol. 8, no. 9, pp. 439–517, 1993.

[31] (https://stackoverflow.com/users/5152701/john) John: Most accurate timer in .net. Stack Overflow, 2015. Accessed: 01-02-2020.
Available at: `https://stackoverflow.com/a/31612770`

[32] Duane, C.B.: Close-range camera calibration. *Photogramm. Eng*, vol. 37, no. 8, pp. 855–866, 1971.

[33] Calitz, N.: *The Design and Implementation of a Stellar Gyroscope for Accurate Angular Rate Estimation on CubeSats.* Master's thesis, Stellenbosch University, 2015.

[34] Sun, T., Xing, F. and You, Z.: Optical system error analysis and calibration method of high-accuracy star trackers. *Sensors*, vol. 13, no. 4, pp. 4598–4623, 2013.

[35] Dzamba, T. and Enright, J.: Calibration techniques for low-cost star trackers. 2009.

Appendices

Appendix A

Star Projection GUI

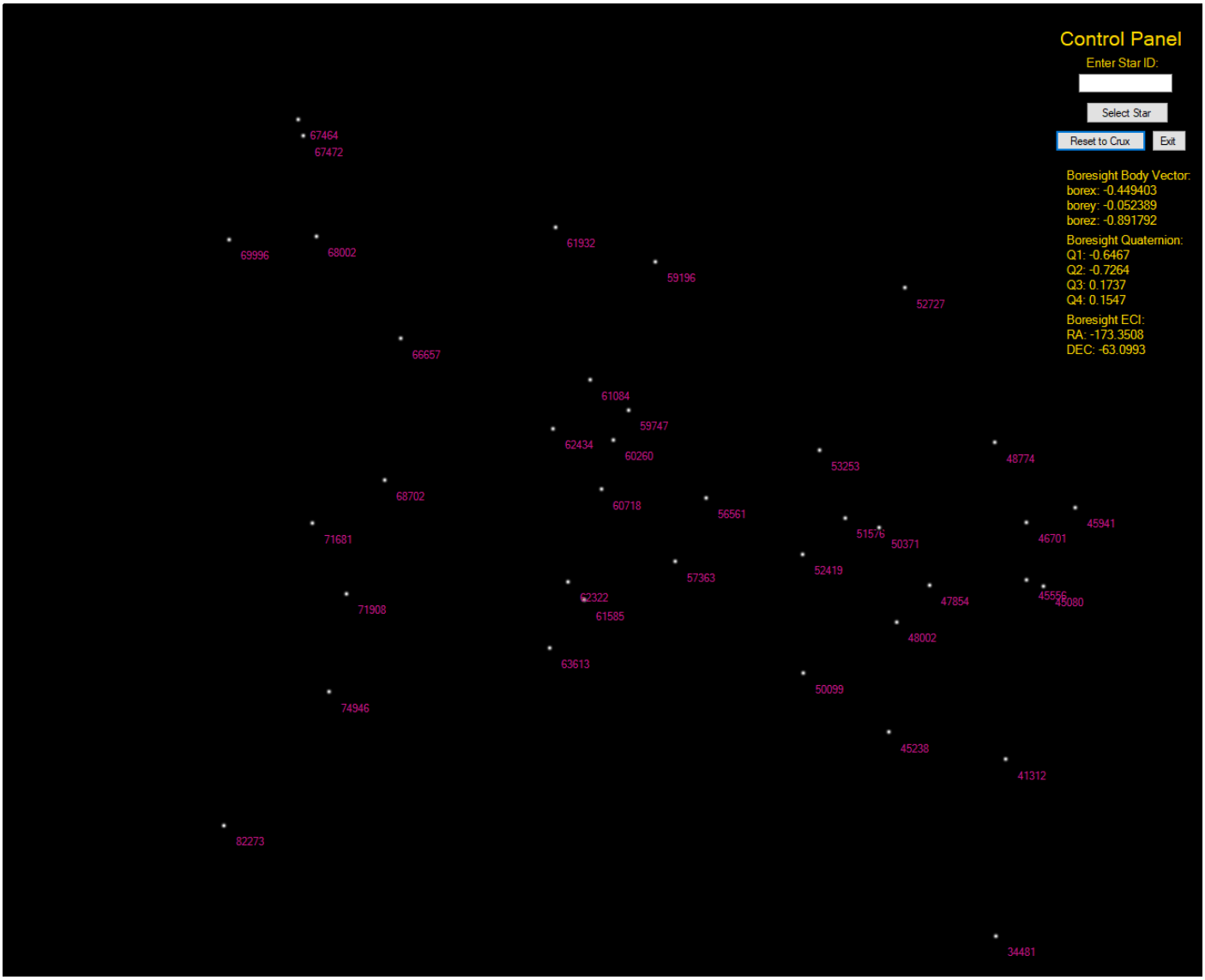


Figure A.1: Star Projection GUI

Appendix B

Star Kernel Generation Results

Table B.1: Resulting Star Kernel Accuracies With Varying Gaussian Distribution Parameters

	Kernel Size:	5x5		7x7		9x9	
Resolution	Standard Deviation	Maximum Error [pixels]	% Pixels Below Threshold	Maximum Error [pixels]	% Pixels Below Threshold	Maximum Error [pixels]	% Pixels Below Threshold
10	0.5	0.0513	64				
10	0.6	0.0574	16				
10	0.7	0.0728	16				
10	0.8	0.1019	0	0.0554	40.82	0.0503	64.1975
10	0.9	0.1355	0	0.0640	24.49	0.0514	54.3210
10	1.0	0.1687	0	0.0769	8.16	0.0544	44.4444
10	1.1	0.2003	0	0.0979	0.00	0.0600	24.6914
100	0.5	0.0027	64				
100	0.6	0.0154	16				
100	0.7	0.0423	16				
100	0.8	0.0797	0	0.0103	40.82	0.0006	64.1975
100	0.9	0.1213	0	0.0252	24.49	0.0027	54.3210
100	1.0	0.1623	0	0.0470	8.16	0.0081	44.4444
100	1.1	0.2003	0	0.0736	0	0.0177	24.6914
1000	0.5	0.0027	64				
1000	0.6	0.0154	16				
1000	0.7	0.0423	16				
1000	0.8	0.0797	0	0.0103	40.82	0.0006	64.1975
1000	0.9	0.1213	0	0.0252	24.49	0.0027	54.3210
1000	1.0	0.1623	0	0.0470	8.16	0.0081	44.4444
1000	1.1	0.2003	0	0.0736	0	0.0177	24.6914